HAPTIC CHARACTERIZATION OF HUMAN SKIN IN VIVO IN RESPONSE TO SHOWER GEL TREATMENTS USING A MAGNETIC LEVITATION DEVICE

by

Ryan T. Yardley

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mechanical Engineering

The University of Utah

December 2012

The University of Utah Graduate School

STATEMENT OF THESIS APPROVAL

The following faculty members served as the supervisory committee chair and members for the thesis of Ryan T. Yardley.

Dates at right indicate the members' approval of the thesis.

Stephen Mascaro, Chair	October 24, 2012 Date Approved
Sanford Meek, Member	October 24, 2012 Date Approved
Jake Abbott, Member	October 24, 2012 Date Approved

The thesis has also been approved by Timothy Ameel, Chair of the Department of Mechanical Engineering and by Charles A. Wight, Dean of The Graduate School.

ABSTRACT

Maintaining healthy skin is vital for protection, sensation, and water conservation in the human body. Skin health is usually judged subjectively through sight and touch, and skin haptic properties are often described using arbitrary terms such as ‘softness’, ‘smoothness’, or ‘dryness’. Although qualitative haptic characterization is often sufficient for general comparison between skin states, a quantitative approach is needed to characterize minute changes in skin condition.

This thesis details the use of a precision magnetic levitation haptic device to characterize the haptic properties of human skin in vivo before, during, and after treatment with each of eight commercially available shower gels. Although prior research has sought to characterize the haptic properties of human skin in vitro and in vivo, very few studies have compared the haptic effects of commercial skin products having relatively similar formulations. Additionally, related studies have typically utilized simple, low-precision devices and fixtures.

Various metrics were used to characterize the haptic properties of human skin, including friction, dynamic skin stretch, and viscous damping. A hybrid force-position control algorithm was developed to control a precision 6-DOF magnetic levitation robotic device with silicone tactor to touch, stroke, or stretch human skin in vivo. Data were collected from 32 human subjects and analyzed primarily to investigate the effects of

shower gel type on each of the chosen test metrics. Other factors investigated include skin test location, order, and subject age and gender.

Results showed significant differences between the effects of various shower gels, especially after accounting for variance between subjects. Most notably, Olay and Soft Soap Nutri-Serums gels yielded the highest skin coefficients of friction 20 minutes after treatment, indicating higher levels of skin hydration than other gel treatments. Conversely, Sweet Pea gel treatment yielded the lowest skin coefficient of friction. Additionally, when applied to the skin as un-lathered gels, Soft Soap Nutri-Serums and Soft Soap Shea Butter yielded viscous damping constants twice that of other gels, while Dove Go Fresh yielded the lowest. When lathered into foam on skin, Sweet Pea gel yielded the highest viscous damping constant, while Dove Go Fresh again yielded the lowest.

To my wife Dani and daughter Rylie,
whose love, support, and sacrifices have made this work possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

Foremost I would like to express my profound appreciation to Dr. Stephen Mascaro for granting me this opportunity and for his patient guidance, as well as to Colgate PalmOlive for sponsoring this research study. I appreciate the collaboration and expertise of Dr. Aixing Fan, Head of Research & Development at Colgate PalmOlive. I would also like to thank my thesis committee members Dr. Jake Abbott and Dr. Sandy Meek for their input, time, and encouragement. I also want to thank Dr. Hollerbach for the use of his laboratory space and equipment.

I wish to extend a special thanks to Paul Tanner, Anaplastologist at Hunstman Cancer Hospital, for his expertise and extensive collaboration in the design and building of various artificial human fingertip tactors as well as the final tactor design used in this research study. I also wish to express my gratitude to my colleague Tom Grieve for his general expertise, training, and advice, and to my Undergraduate Research Assistant Rachel Ware for her hard work, great ideas, and fortitude through many hundreds of hours of testing. Lastly but certainly not least, I deeply appreciate the support and sacrifices of other friends, family, and many others who have helped to make this research study a success.

CHAPTER 1

INTRODUCTION

Maintaining healthy skin is vital for protection, sensation, and water conservation in the human body. Skin health is usually judged subjectively through sight and touch, and skin haptic properties are often described using arbitrary terms such as ‘softness’, ‘smoothness’, or ‘dryness’. Although qualitative haptic characterization is often sufficient for simple general comparison between skin states, a quantitative approach is needed to characterize minute changes in skin condition. For instance, characterizing the effects of shower gels or other skin products on the haptic properties of human skin requires reliable, repeatable quantitative methods.

1.1 Motivation

This research was privately sponsored by Colgate-Palmolive to quantitatively characterize the haptic properties of human skin in response to treatment with specific shower gels currently on the market. The project sponsor planned to investigate potential correlations between the quantitative data from this project and qualitative data from consumer studies conducted previously by the sponsor. Eight commercially available shower gels were selected for this project, including three gels from SoftSoap, a subsidiary of Colgate-Palmolive.

The goal of this research was to characterize the haptic properties of human skin before, during, and after shower gel treatments.

1.2 Background and Related Work

Skin is the human body's outermost barrier of protection against the external environment and is essential for the body's proper heat regulation, haptic sensation, and water conservation. To maintain good skin health, most people regularly utilize skin products of some kind for cleansing or hydration. Dermatological health is typically monitored qualitatively through haptic sensation or visual inspection, but the utility of qualitative skin assessment is inherently limited by poor resolution and differing individual perception. To compare the effects of similar skin care products, a repeatable quantitative approach is required.

Over the past decades, researchers have discovered a number of effective quantitative methods for characterizing the mechanical and haptic properties of human skin in vivo. Measuring the coefficient of friction of the skin is perhaps the oldest and simplest method of skin characterization as it mimics the common sensation of using one's finger to stroke skin during common qualitative skin inspection. One of the earliest pioneers in skin friction measurement, Naylor showed that moistened skin has higher friction [1]. Shortly after, Comaish and Bottoms showed that human skin does not obey simple friction laws but instead displays a much more complex friction interaction, likely due to its deformation being viscoelastic and not plastic [2].

Skin friction is typically measured through either linear or rotational motion of a probe against the skin. Highley et al. used a rotational friction probe to evaluate the

frictional effects of hydrations, oils, and surfactants on human skin in vivo, and found that friction increases with skin hydration [3]. El Shimi also used a rotating friction probe to confirm the findings of Comaish and Bottoms regarding human skin's deviation from simple friction laws, and also concluded that dry skin yields lower friction than normal skin [4]. Weinstein used a custom friction measurement device composed of a simple mesh sled probe that was pulled across the skin while a transducer measured the friction force [5]. These sled experiments quantified skin smoothness and evaluated different razors based on the closeness of shave. Nacht et al. used the same torsion friction meter as Comaish and Bottoms, but used it to quantify the greasiness of various skin treatments and correlate it with subjective opinion data [6]. Nacht et al. also showed that treatment of the skin with water only caused an immediate increase in skin friction, which rapidly faded back to the skin's steady state friction after only a few minutes. Wolfram used a very simple device to pull a weight of fixed mass across the skin surface by using a falling counterweight to regulate velocity [7]. He concluded that moisturization with a cosmetic product increases skin friction due to a reduction in skin elasticity and protein materials being dissolved by water at the skin surface.

More recent friction studies have typically focused either on exploring novel skin friction measurement methods or better explaining the reasons for human skin's complex friction behavior. Zhang and Mak used a rotational friction meter to measure friction of normal untreated skin in six regions of ten human subjects using five different probe materials. They found that friction was generated from either skin deformation as a "ploughing" effect, or from skin adhesion [8]. Koudine et al. investigated the effects of cosmetic treatments on skin friction using glass skin traction tests with an apparatus

capable of exerting normal loads up to 100 mN at 200 Hz sampling frequency; the tests confirmed that friction increases with cosmetic product treatments [9]. Asserin et al. used a custom device to show that cutaneous smoothness is an effect of both skin friction and mechanical properties [10].

Many of the skin measurement studies from the past decade have branched out to include not just friction, but other skin metrics as well to better characterize human skin. Gitis and Sivamani thoroughly summarized the efforts of related research and introduced a device which simultaneously measures both friction and electrical properties of skin such as resistance and capacitance [11]. Ramalho et al. used sliding friction tests with uncontrolled normal force and Trans-Epidermal Water Loss (TEWL) measurements to characterize hand and forearm skin in vivo in response to the moisturizer ointments petrolatum and glycerin [12]. They found that skin friction varies with anatomical site and that in vivo forearm skin friction increases with time after treatment with moisturizers. Pailler-Mattei et al. used progressive tape-stripping to isolate the contribution of the stratum corneum while using indentation and friction tests to measure skin elasticity, adhesion, and friction properties [13]. Liu et al. explored how haptic perception is affected by skin surface topography and friction [14]. They measured friction and contact forces of fingertip stroking on various materials and compared the quantitative results with subjective human perception data. Kwiatkowska et al. used a reciprocating-sliding test with an optical sensor and camera to study the "bow-wave" of the skin formed during sliding as a consequence of friction and lateral skin displacement [15]. From the sliding tests, the stick and slip phases of friction force were identified and used to determine effective skin stiffness, which was compared with effective indentation

stiffness. Zahouani et al. used a new bio-tribometer for traction and indentation tests to measure skin friction and elasticity [16]. Zahouani et al. used the friction noise from a novel tribo-acoustical probe to give an indication of skin softness, and showed that both adhesion and contact area are major contributors to friction force [17].

Besides skin friction, skin indentation has been the second most common skin measurement metric. In fact, prior to 2004, very little skin indentation research was pursued with the exception of Weinstein, who supplemented his skin friction experiments with extensive indentation experiments for quantifying changes in skin softness [5]. More recently, Pailler-Mattei and Zahouani have dominated recent advances in skin characterization using indentation. They used a custom indentation device to measure the adhesion of the skin surface during indentation unloading. The skin adhesion measurements were used in conjunction with advanced surface contact theory to quantify changes in the lipid film of the skin surface during hydration with water [18], [19] . Delalleau et al. also used skin adhesive properties from indentation tests to study the effects of skin aging [20]. Boyer et al. used dynamic micro-indentation to characterize the dynamic properties of skin at frequencies up to 60 Hz [21]. Pailler-Mattei, Bec, and Zahouani used indentation and multilayer models to estimate the Young's modulus of the dermis layer of human skin [22]. They concluded that it is necessary to account for subcutaneous layer properties in order to use indentation to accurately characterize the dermis layer.

Other less common skin measurement metrics have also been used to characterize human skin properties with varied success. Agache et al. used the angular skin deflection under a constant applied torque to estimate Young’s modulus and other mechanical

properties of skin [23]. Kajs and Garstein reviewed a number of uncommon but potentially useful skin measurement metrics, including water retention, color, blood microcirculation, viscoelastic properties, surface profile, and desquamation [24]. Hendriks et al. used suction with ultrasound and optical coherence tomography to determine the mechanical properties of different layers of human skin[25]. A different suction aperture size and length scale was used to isolate the mechanical response of each layer of skin.

This research focuses on using active force and position control of a robotic device to characterize changes in human skin properties before, during, and after treatment with shower gel products. Selected metrics include friction, viscous damping, and dynamic skin stretch. Similar to most related skin characterization studies, the volar forearm was selected as the skin test location because of its convenience, relative flatness, and minimal body hair [4], [10], [11], [15], [16], [26]. Based on this literature review, it is believed that no other studies have used a robotic device employing magnetic levitation to measure skin properties. Additionally, it is believed that no other studies have attempted to quantitatively measure minute changes in skin properties before, during, and after treatments with potentially similar formulations of commercially available shower gels. This research study evaluates the effectiveness of this novel approach.

CHAPTER 2

METHODS

2.1 Experimental Setup

2.1.1 Physical Setup

The principal machine used in this study is the Magnetic Levitation Haptic Device (MLHD) from Butterfly Haptics, as shown in Figure 2.1. The MLHD is spherical in shape and uses a series of 6 Lorentz actuator coils to levitate an untethered central component, or 'flotor'. Since the flotor has no physical contact with the rest of the device, the MLHD has no friction except to air resistance and also has no mechanical backlash. The flotor has a spherical workspace of 12mm radius, a position precision of less than 2 microns, and a position bandwidth of >140 Hz. The MLHD is capable of exerting 3D forces up to 40 Newtons with a force resolution of 20 mN and a force bandwidth of >2000 Hz. The MLHD's maximum force is not large enough to pose a threat to human subjects. Additionally, its high precision makes it ideal for the exact position and force trajectories required for haptic characterization. The MLHD utilizes internal closed-loop position control to achieve desired position trajectories. To achieve desired force trajectories, only open-loop force estimation is used, although there is no reason why MLHD users cannot implement closed-loop force control with a third-party sensor.

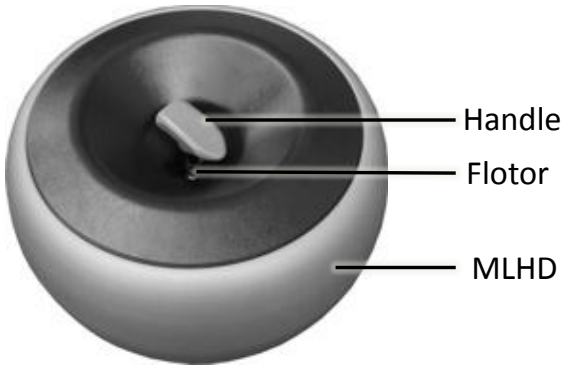

Figure 2.1 Original MLHD from Butterfly Haptics

To permit closed-loop force control, the MLHD flotor was modified to include an ATI Nano-17 6-axis force/torque sensor. The Nano-17 sensor was mounted to the top of the flotor to replace the original user hand interface. The new flotor assembly is shown in Figure 2.2. Data from the force sensor were collected by a Sensoray 626 data acquisition card, which was connected to a client computer running Linux Ubuntu. Figure 2.3 shows the MLHD client server interface. The client computer also interfaces via an Ethernet cable with the MLHD server computer, which runs QNX. The server computer interprets and relays control commands to the MLHD and returns MLHD information to the client computer upon request. Control code was written and run in C++ on the client computer, and collected data was stored in delimited text files on the client computer.

A custom tactor was attached to the top of the Nano-17 sensor to serve as the system end-effector. Originally the custom tactor was designed as a compliant artificial human fingertip, including multiple layers of silicone or polypropylene, but the compliant designs were abandoned since they damped high-frequency vibration. To minimize the loss of high-frequency information collected by the force sensor, a new tactor was constructed using a steel cap nut coated in a thin layer of NuSil Med 4014 silicone for

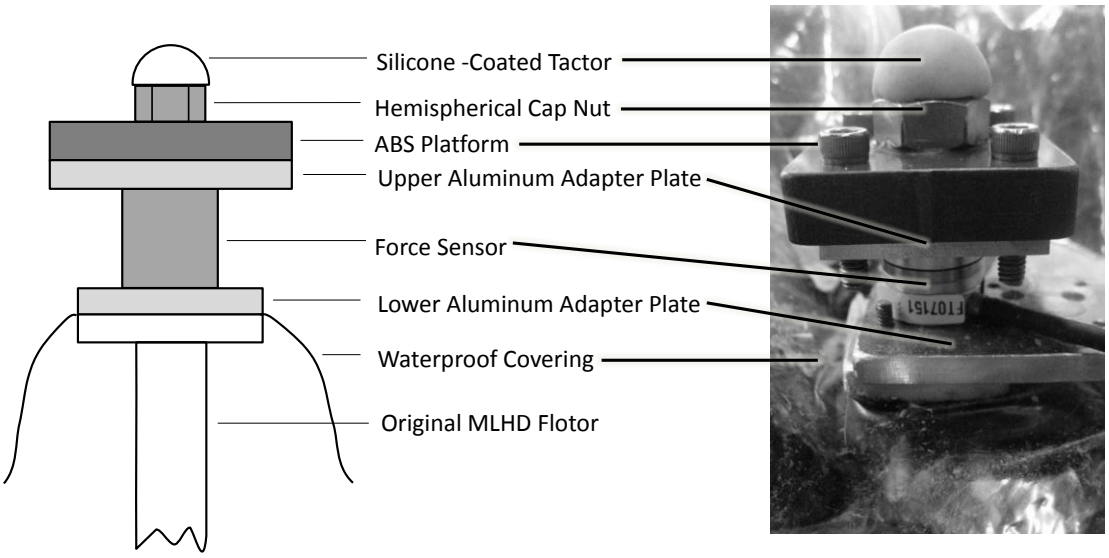

Figure 2.2 Magnetic Levitation Haptic Device modified flotor and tactor

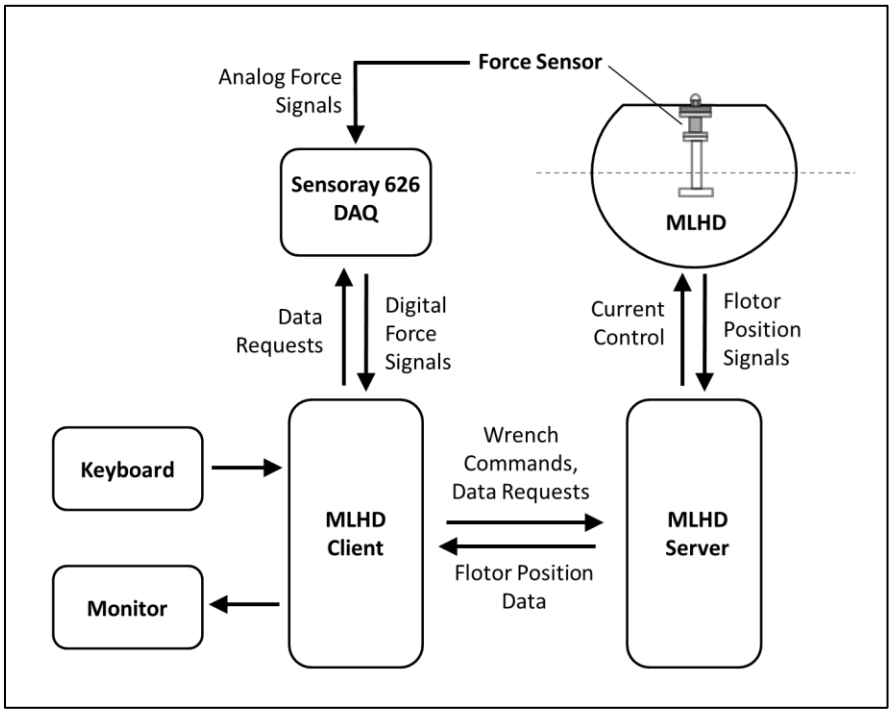

Figure 2.3 MLHD client server interface

increased friction. The original hemispherical cap nut was 15.875 mm in diameter, and its diameter increased by approximately 1 mm with the addition of the silicone coating. The silicone-coated cap nut was then epoxied to an ABS plastic platform so that the hemispherical end of the cap nut faced upward and provided a relatively uniform contact surface given reasonable arm positioning, regardless of skin surface topography.

To protect the MLHD from likely moisture and shower gel exposure during experiments, a waterproof protective covering was added. Although various materials and complex solutions were considered, a 0.127 mm thick sheet of flexible transparent plastic was chosen for its low mass to minimize changes to the flotor dynamics. The sheet was inserted between the top of the MLHD flotor and the bottom of the force sensor, and rubber O-rings were used to seal the small holes which were cut from center of the sheet for the sensor fasteners.

The volar forearm was chosen as the primary skin test location for its convenience, general lack of body hair, and relatively planar surface. Since the MLHD was designed to be mounted statically in a table with the flotor aligned vertically with the gravity vector, it was decided that test subjects' volar forearms would be tested face down over the MLHD. Additionally, although the volar forearm represents a large test area compared to the MLHD's limited workspace, for procedural convenience it was determined that each volar forearm could contain no more than two test areas.

To ensure that haptic test results were indicative of changes in skin haptic properties and not simply changes in test location, a procedure was devised to mark the skin test locations accurately and in a repeatable manner. First, the test subject was asked to raise one arm, with the upper arm extended horizontally and the forearm and fingers pointing

vertically, and the palm of the hand facing. The test administrator then located the Medial Epicondial of the Humerus bone near the elbow, as well as the Pisiform bone near the palm of the hand. A graduated ruler and a washable ink pen were used to mark a dotted straight line between the two landmarks beginning at the midpoint and extending posteriorly toward the Medial Epicondial of the Humerus bone, as shown in Figure 2.4. A stencil was used to mark two ovals on each arm indicating the skin test locations to be used. The straight dotted line was used to represent the lengthwise central axis of the test location ovals. The stencil test location ovals were 20 mm in length and 10 mm in width and end diameter to accommodate the dimensions of the tactor diameter and the maximum planned stroke length of 16mm. The oval test locations were separated by a

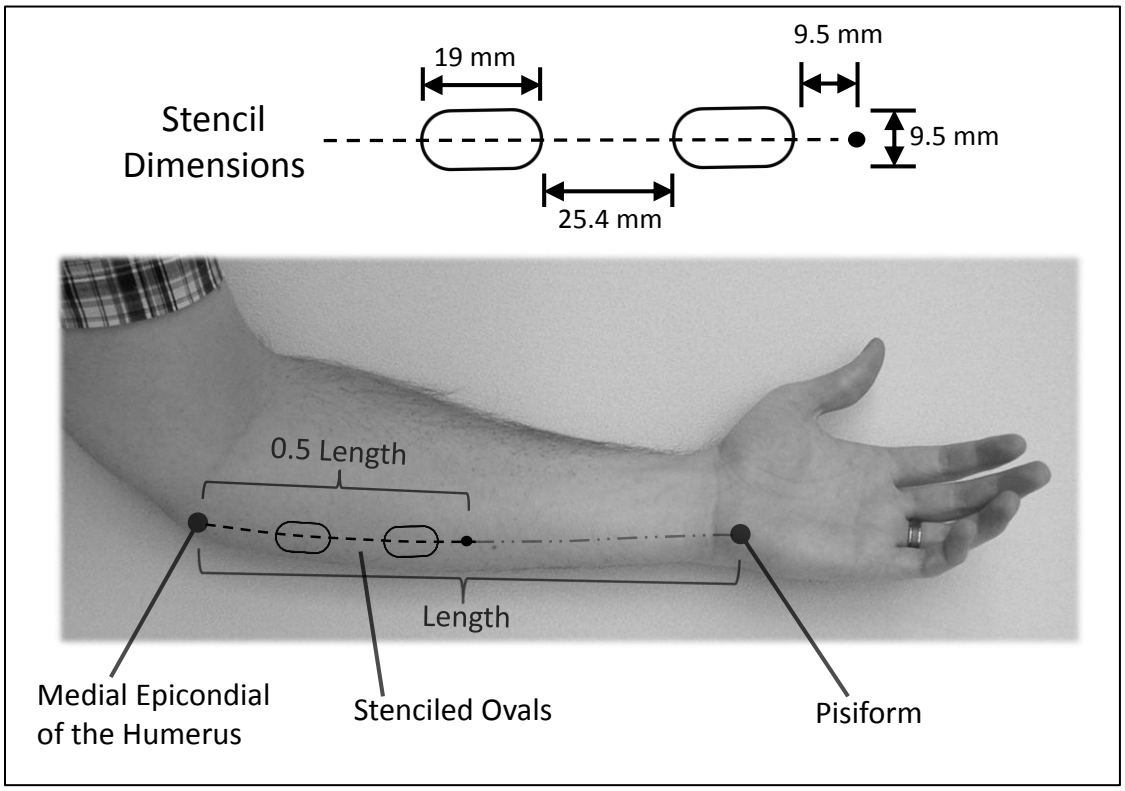

Figure 2.4 Skin test location marking method and measurements

center-to-center distance of 45 mm to maximize the potential similarity of skin between test locations while still permitting enough separation to minimize cross-contamination of gels between test locations during experiments. The test location marking procedure was repeated on the other arm of the test subject. Test locations were labeled with numbers one through four, left to right and distal to proximal, beginning with the left distal location and ending with the right proximal location, as shown in Figure 2.5. The washable ink was considered to have an insignificant effect on skin properties, but was used and reapplied sparingly nonetheless.

Because the custom tactor extended beyond the upper edge of the MLHD, and due to the limited workspace of the MLHD flotor, an adjustable platform was constructed to position and support test subjects' arms. The platform was constructed using an aluminum plate of dimensions 31" length by 6" width by 0.25" thickness. A 3" diameter hole was cut in the center and positioned over the center of the MLHD to allow the tactor to access the test subject arm. The hole was designed with a safety factor to avoid contact interference with the flotor during rare periods of instability. An opposing design consideration was that a larger hole would provide less arm support and would allow arms with a large curvature to hang below the bottom of the armrest plate in an inconsistent manner. A 3" hole was determined to be the best compromise between skin surface positioning, test subject comfort, and flotor safety during instability. The plate was covered with neoprene padding and a thin layer of waterproof imitation leather for test subject comfort and easy armrest cleaning. The armrest platform accommodated both right and left arms seamlessly, without any need for time-consuming adjustments when switching arms.

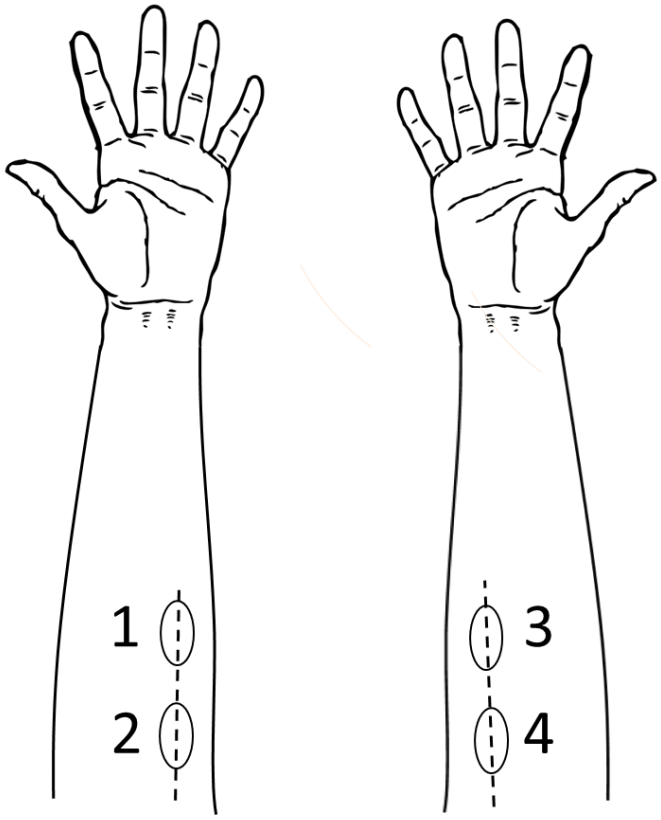

Figure 2.5 Numbering of skin test locations

Due to the small size of the platform hole, average arms typically completely covered the hole, making it difficult for test subjects to accurately place the test location oval directly over the tactor. To remedy this, a miniature pinhole camera was attached to the underside of the arm rest platform, and pointed and focused at an upward angle toward the usual skin/tactor interface location. An array of three white LEDs was used to improve lighting conditions under the armrest, which were otherwise very poor when the hole was covered by a test subject's arm. The camera was connected to a small display screen, which test subjects could use to aid in centering test location over the tactor during arm positioning.

To aid in the relaxation and consistent positioning of test subjects' arms, an ergonomic hand rest was constructed from soft foam. It was sized and shaped to mimic a common computer mouse, and was covered in soft cotton fabric. In early preliminary testing, it was observed that without some standard ergonomic hand rest, test subjects' random hand placements typically changed muscle tension and skin stretch, often resulting in greater noise. The ergonomic hand rest is positioned freely on top of the arm rest platform, and it successfully minimizes arm flex and improves subject comfort.

The covered plate was supported by two adjustable Jiffy-Jack laboratory jacks from Cole-Parmer. Each jack was rated for 132 lb static load, which was well beyond the expected static weight of any test subject's arm, and thus suitable for adjustment even while loaded. Each jack was height-adjustable from 2" up to 11", which was a much greater range than needed. The jacks permitted fine height adjustment of the armrest platform, which allowed the test administrator to position the subject's forearm vertically to lay well within the MLHD workspace limits. Additionally, this method of precision arm positioning allowed for simplified control algorithms since the MLHD could begin in active force control with the tactor already exerting some known force on the test subject's skin. Otherwise, the MLHD would have had to begin in position control, find the skin surface, and then switch to force control. Figure 2.6 depicts the physical setup design and Figures 2.7 through 2.10 show elements of the actual test setup.

2.1.2 Control Algorithm

A hybrid force/position controller was implemented to control the MLHD's 6 degrees of freedom simultaneously using either closed-loop force control or closed-loop position

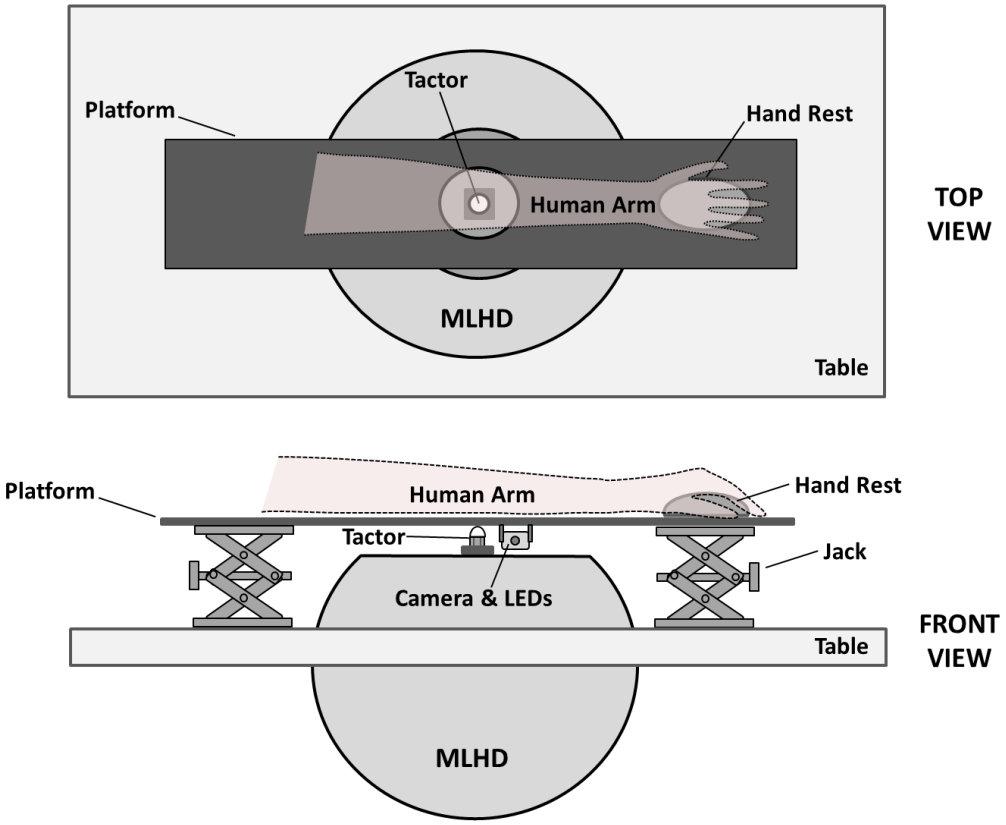

Figure 2.6 Physical experimental setup design

Figure 2.7 Actual physical experimental setup

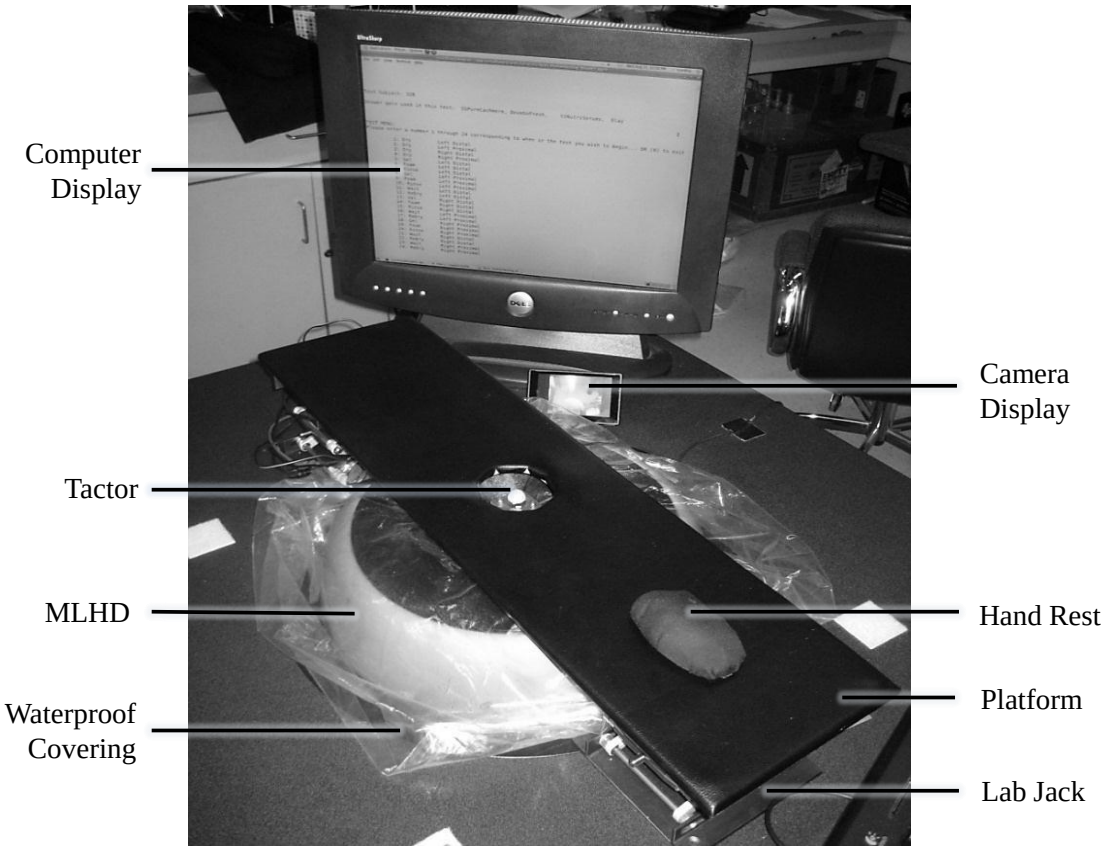

Figure 2.8 Close-up view of arm rest

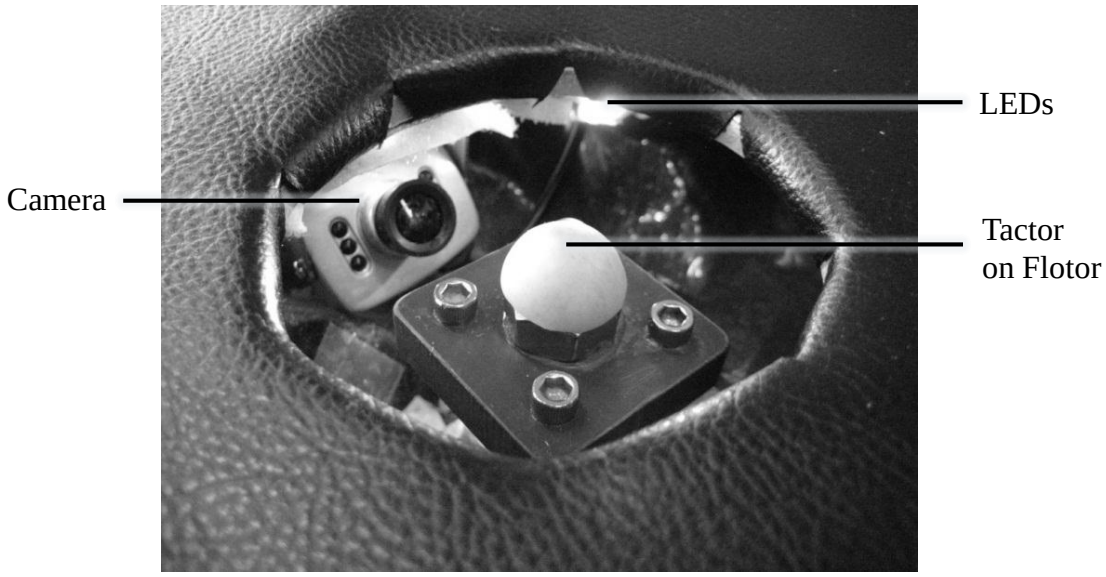

Figure 2.9 Close-up view of tactor, camera, and LED lighting

Figure 2.10 Close-up view of laboratory jack and hand rest

control [27]. Hybrid force/position control can be used to maintain a constant force normal to a surface using force control while simultaneously moving tangentially along the surface using position control, making it an especially useful controller for friction measurement of curved compliant surfaces like the human arm. The default MLHD internal controller was of limited use since it relied solely on closed-loop position control and open-loop force estimation. By gradually decreasing the internal MLHD gains to zero while simultaneously increasing the new hybrid controller gains to their full target values, the new hybrid controller was able to stably override the internal MLHD controller. The Hybrid controller could then actively utilize the MLHD's position data as well as the force/torque data from the Nano-17 sensor. Additionally, because the Jacobian and Inertia matrices are effectively built into the MLHD communications interface, all axes were fully decoupled so that only a wrench was required for direct control of the flotor. Since Jacobian operations and the Inertia Matrix are accounted for by the MLHD server, the hybrid position/force control block diagram was significantly simplified, as

shown in Figure 2.11. Note that X represents a position vector, F represents a force vector, W represents a wrench, and S represents the selection matrix. Additionally, R represents a frame rotation from subscript frame to superscript frame, where C, 0, and T represent the constraint, zero, and tool frames, respectively.

A PIV controller was designed to control both force and position in their respective active degrees of freedom. To select which axes were controlled by force and which by position, a 1 by 6 selection vector was used to populate the diagonal of the hybrid control selection matrix. The selection vector or diagonal of the selection matrix contained either ones or zeros, with ones indicating the degrees of freedom to be controlled using position control, and zeros indicating force control. The 6 by 6 selection matrix and was multiplied by the 6 by 6 gains matrix to form the position control gains matrix, and the difference between Identity and the selection matrix was multiplied by the gains matrix to yield the force control gains matrix. Rotation matrices were used to transform between the Nano-17 sensor frame and the surface frame. Due to the difficulty involved in identifying actual MLHD system parameters, the external controller gains could not be chosen directly through calculation; instead, the gains were tuned experimentally. The hybrid controller operating frequency was set at 1000 Hz.

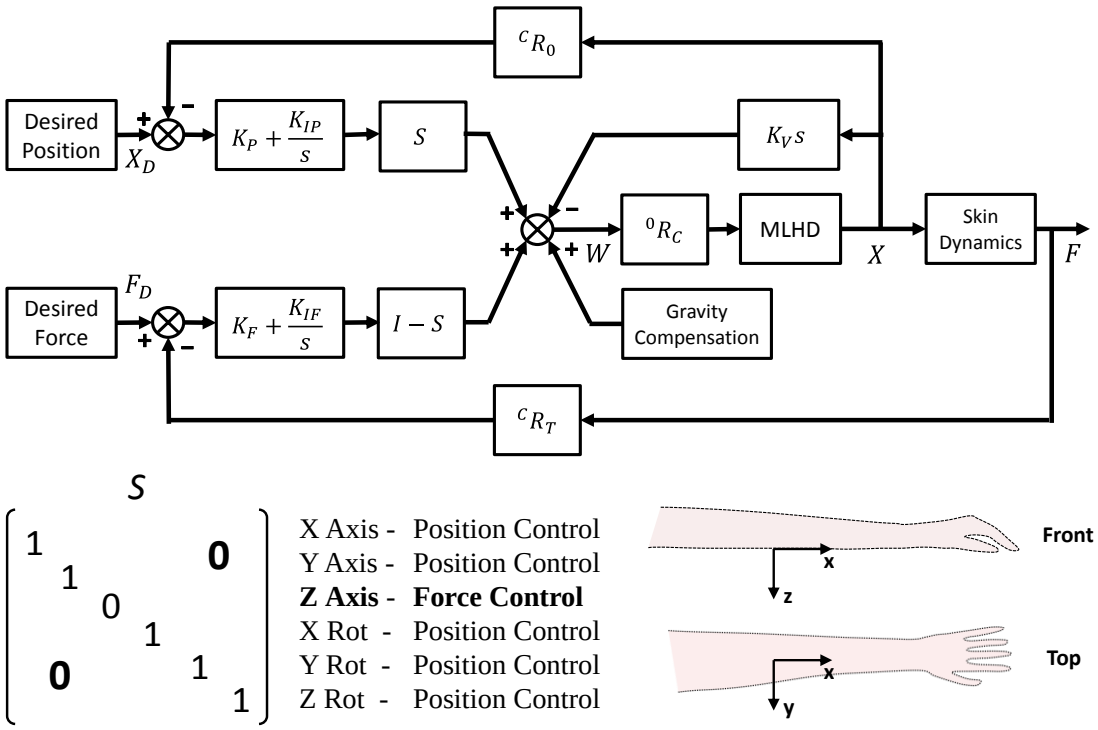

Figure 2.11 Hybrid position/force controller diagram, selection matrix, and arm coordinate system

2.2 Experimental Design

2.2.1 Test Types and Metrics

2.2.1.1 Friction

For quasi-static dry skin haptic characterization, friction was selected as the primary metric because of its proven reliability as an indirect measure of skin hydration. The hybrid force/position controller was especially useful for determining friction coefficients of human skin since a constant normal force could be maintained while simultaneously stroking the curved compliant skin surface tangentially at constant velocity. The normal force exerted by the tactor on the skin was controlled to be only 0.1 N to minimize

interference effects from subcutaneous tissue while still maintaining good stability. In order to minimize occurrence of the instabilities caused by surpassing the MLHD workspace boundaries, the tactor was programmed to travel only 16 mm of the available 24mm during each stroke. A stroking velocity of 2 mm/s was chosen arbitrarily as a reasonable compromise between minimizing stroking time and maximizing data resolution through slower stroking. During each stroking test, the tactor stroked the skin test location with a repeating reciprocal action for a total of 10 strokes, with 5 strokes in each direction along the MLHD x-axis, parallel to the forearm of the test subject. All time, force, and position data were sampled at 1000 Hz and stored in delimited text files to be accessed later for analysis. This test was performed every time a skin location was tested before or after treatment with shower gel.

2.2.1.2 Dynamic Skin Stretch

To characterize dynamic haptic skin response, system identification techniques were employed to determine mass, damping, and stiffness constants of the tested skin location. The MLHD flotor was controlled to maintain a constant normal force while following a sinusoidal position trajectory tangential to the skin surface. To minimize skin slip, the amplitude of the sinusoidal trajectory was set at 0.5 mm, while a constant normal force was maintained at 0.5 N. A series of 26 frequencies was used to vary the desired position, with each unique sinusoidal frequency trajectory temporally separated by a 1 second pause to allow the MLHD flotor to settle. The frequencies were initially spaced equally on a logarithmic scale between 5 and 150 rad/sec, but more frequencies were added to increase resolution near the estimated resonant frequency. Table 2.1 lists all 26

Table 2.1 Dynamic Skin Stretch frequencies (rad/sec)

5.0000	42.6016	53.1139	57.4667	75.1145
8.4090	45.3725	53.9573	58.3792	80.0000
14.1421	48.3236	54.8140	62.1763	93.6139
23.7841	51.4666	55.6844	66.2203	109.5445
40.0000	52.2838	56.5685	70.5273	128.1861
				150.0000

frequencies. The actual position of the tactor and the actual skin force response were sampled at 1000 Hz and recorded in text files for later reference.

In post-processing, the amplitude ratio and phase shift were calculated for each frequency using fast Fourier transform methods. To isolate the skin dynamics from the MLHD flotor dynamics, the amplitude of the actual tactor position and the amplitude of the actual measured force response of the skin were identified as the input and output of the amplitude ratio, respectively. A Bode plot was generated, and a second-order transfer function was estimated from the Bode plot. The mass, damping, and stiffness constants of the skin were determined from the transfer function. This test was conducted on each test location before and after shower gel treatment.

2.2.1.3 Viscous Damping

Viscous damping was also selected as a metric for the haptic characterization of the gel/skin and foam/skin interfaces during shower gel treatment and lathering. A constant normal force of 0.1 N was applied while maintaining a constant tangential velocity. The tactor stroked the skin for two strokes and a distance of 16 mm per stroke, initially at a velocity of 4 mm/s. The velocity doubled every two strokes thereafter for four additional

strokes, for a total of three velocities over six strokes. Figure 2.12 shows a conceptual plot of the desired position trajectory during the viscous damping test. The tangential force was recorded and the average slip force was determined for each velocity. The average slip force was plotted versus velocity, and a line was fit through the three points. If the linear fit had a correlation coefficient greater than 0.9, the slope of the line was accepted as the viscous damping constant for the gel/skin or foam/skin interface in question. This test was only conducted during treatment with shower gels.

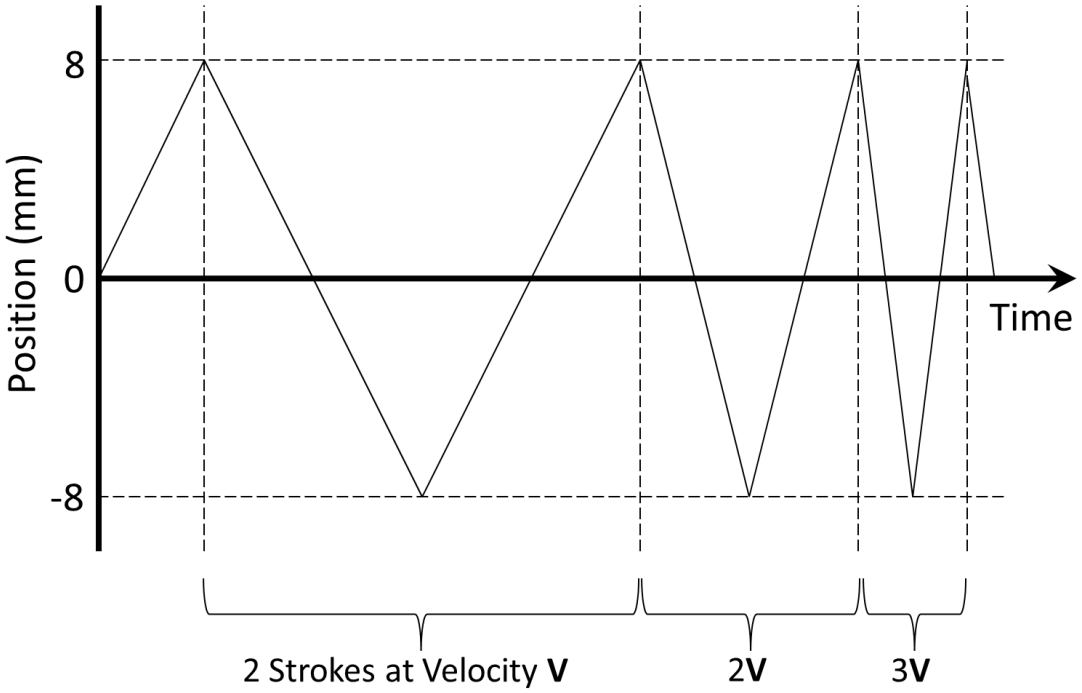

Figure 2.12 Desired position trajectory of viscous damping test

Stroking velocity increases every two stroked to measure resulting tangential force at various velocities.

2.2.2 Experimental Procedures

Since the experiment focused on characterizing human skin haptic properties before, during and after shower gel treatments, test procedures were made to simulate common showering conditions as much as possible in a laboratory environment. The most important of these conditions was considered to be the wetting of the skin with water. Hot water was selected based on the assumption that most people in the sponsor's target market use hot water instead of cold or room-temperature water when showering. Additionally, early preliminary testing strongly indicated that use of hot water provided much better data repeatability than room-temperature water during shower gel treatment.

One of the greatest challenges encountered in early preliminary testing was how to consistently regulate how much water was applied to the skin test locations during shower gel treatment. A container of hot water was maintained at 50°±2° C to soak and heat clean washcloths. When preparing a skin test location for subsequent shower gel application, instead of soaking or pouring hot water over the skin test location in an uncontrolled manner, the hot wet washcloths were held against the skin test location for a specific number of seconds. The heated wet washcloths were assumed to hold a relatively equal volume of water, and allowed heat and water to be applied for an extended period of time to a specific skin location without the need for the subject's entire arm or body to be exposed to heat and water.

2.2.2.1 Detailed Test Procedure

The test subject signed the consent form, experiment instructions were given by the test administrator, and the skin test locations were marked. The test administrator ran a

computer program that gave step-by-step instructions and timing information throughout the experiment. The test administrator instructed the test subject to place the left arm face down onto the arm rest, parallel with the length of the arm rest, with the location 1 oval (left distal volar forearm) centered over the MLHD tactor. The test subject was instructed to use the camera display to aid in test location positioning and to relax the hand over the ergonomic hand rest. An example of the camera display is shown in Figure 2.13. This same arm positioning procedure was repeated before each experiment.

The computer program was continued, giving an initial real-time force readout on the computer screen for 10 seconds which allowed the test administrator to finely adjust the height of the armrest until the target normal force of 0.1 N was reached; reaching the initial target normal force also indicated that the skin surface was approximately centered vertically in the limited workspace of the MLHD tactor, thereby minimizing chances for instabilities due to exceeding workspace limits later in the experiment. The test administrator then initiated the stroking and dynamic skin stretch tests to characterize the original dry skin haptic properties. As instructed step by step by the computer program, the initial dry skin test was then repeated on each of the remaining three skin test locations. A hot wet washcloth was then held firmly against Location 1 for 60 seconds by the test subject, without allowing water to drip onto other test locations, as shown in Figure 2.14. After the computer program timer completed the 60 second countdown, the washcloth was removed. The test administrator applied 0.5 mL of shower gel to Location 1 using a graduated syringe without a needle as shown in Figure 2.15, and the viscous damping test was performed.

The test subject was then instructed to use a wrung-out hot washcloth to wipe once

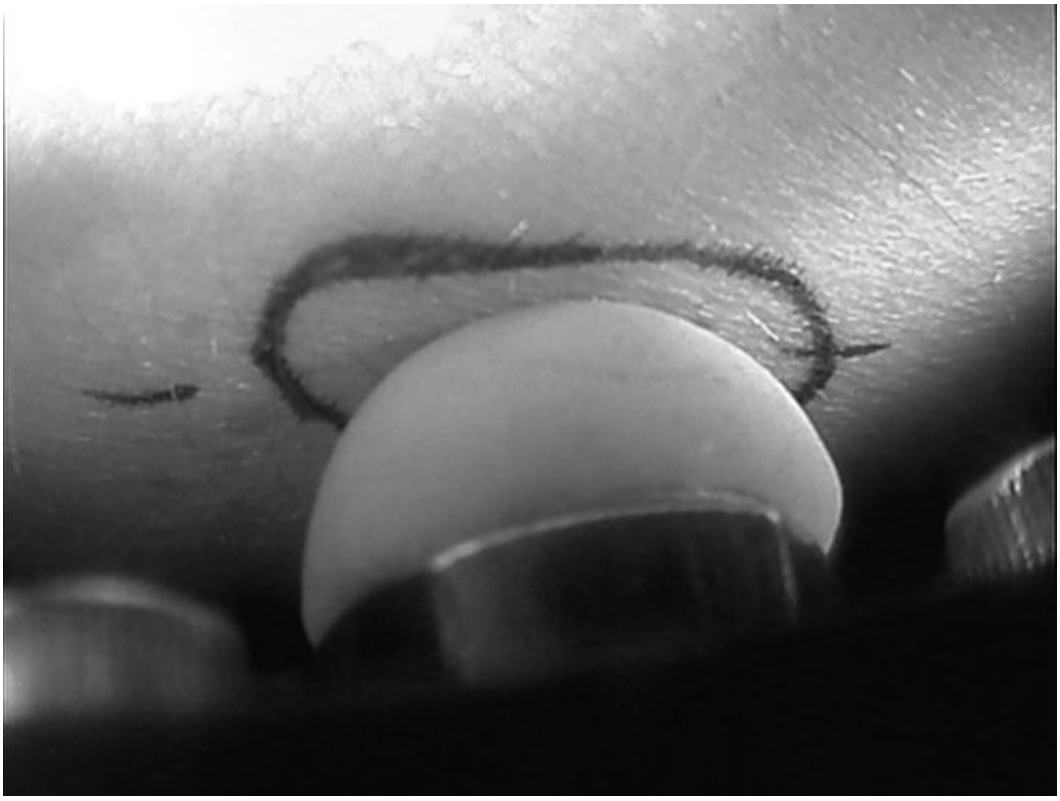

Figure 2.13 Camera view of tactor stroking a skin test location

Figure 2.14 Hot wet wash cloth resting on skin test location

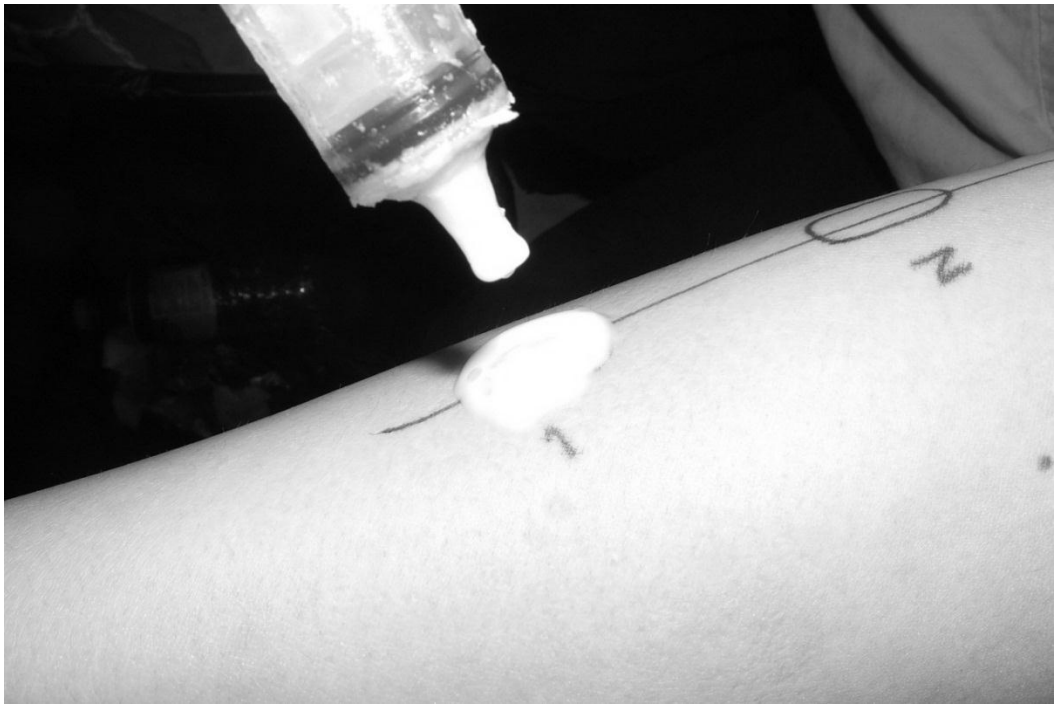

Figure 2.15 Applying shower gel

across Location 1 to remove the gel, while the test administrator thoroughly cleaned the tactor repeatedly with water and paper towels. If the gel were not removed from both the skin and the tactor, an inconsistent amount of gel would be allowed to remain and perhaps affect the test results. The test administrator then dampened a small sea foam sponge (approximately 8 cm^3 in volume, and prismatic in shape) and applied 1 mL of the current shower gel to the sponge. The test subject was instructed to lather Location 1 for 30 seconds by repeatedly squeezing the sponge and rubbing the skin in a circular pattern with the sponge to generate as much foam as possible, as shown in Figure 2.16.

Location 1 was again tested using the viscous damping test. The test subject was instructed to use the same damp washcloth used previously to wipe Location 1 three times with a clean area of the cloth each time to remove the foam while the test

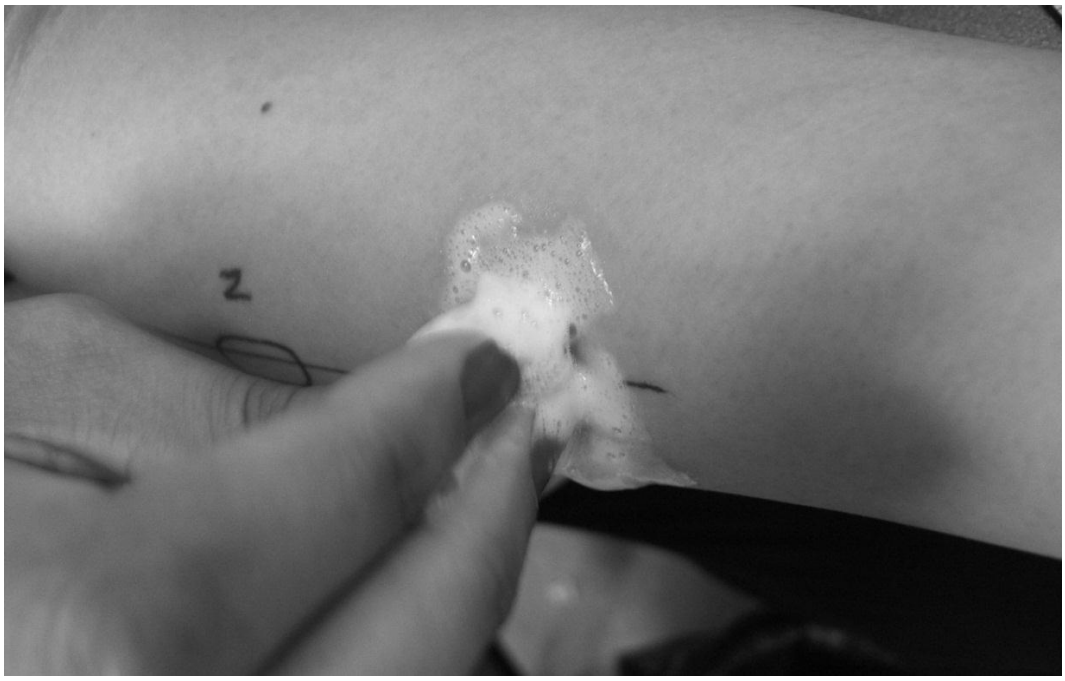

Figure 2.16 Lathering gel into foam

administrator cleaned the tactor thoroughly. A clean dry washcloth was used to pat the skin dry. If it was close to fading completely, the oval marking was redrawn.

Location 1 was tested again using the dry stroking and dynamic skin stretch tests, and the computer initiated a unique 20-minute timer to run in the background while the shower gel was applied to the next skin test location in the same pattern. After 20 minutes had elapsed, the computer program instructed that Location 1 be retested one final time using the dry stroking and dynamic skin stretch tests. The entire process was repeated for all four areas, as mediated by the computer program's timing and instructions. When necessary, the test administrator and test subject were instructed to wait until the 20 minute timer had completed before moving on to the next location, affording the test subject a chance to stretch or take a break. Each test session typically lasted about 1 hour

and 45 minutes. A second test session was also required for each test subject in order to test the remaining four shower gels.

2.2.3 Analysis Methods

2.2.3.1 Friction

Figure 2.17 shows a plot of the tangential force and position data versus time from a typical ten-stroke friction test. The data appear as a modified square wave of alternating vertical orientation due to the reciprocating tactor motion, with positive changes in tactor position resulting in negative changes in tangential force, and vice versa. Figure 2.18 shows the same force data plot zoomed in on a single stroke, with important sections of the stroke labeled.

When the tactor begins moving, the friction and adhesion forces cause the skin to stick to the tactor and begin to stretch as the measured tangential force builds, as shown by the sloped section of the data. When the force is great enough to overcome the friction and adhesive forces of the skin, slip occurs and the measured tangential force levels out to a constant, as shown by the zero-slope section of the data. During this slip phase, the tactor slides along the skin, and the force measured is used to calculate the skin friction coefficient of that location.

A piecewise linear regression method was used to determine the slip point, or the point at which the tactor begins to slip across the skin with a relatively constant tangential force. A MATLAB add-in for constrained linear regression using Shape Language Modeling [28] was used for piecewise linear regression of the friction data. Two lines were used with three knots, with the first line constrained to maintain a positive slope,

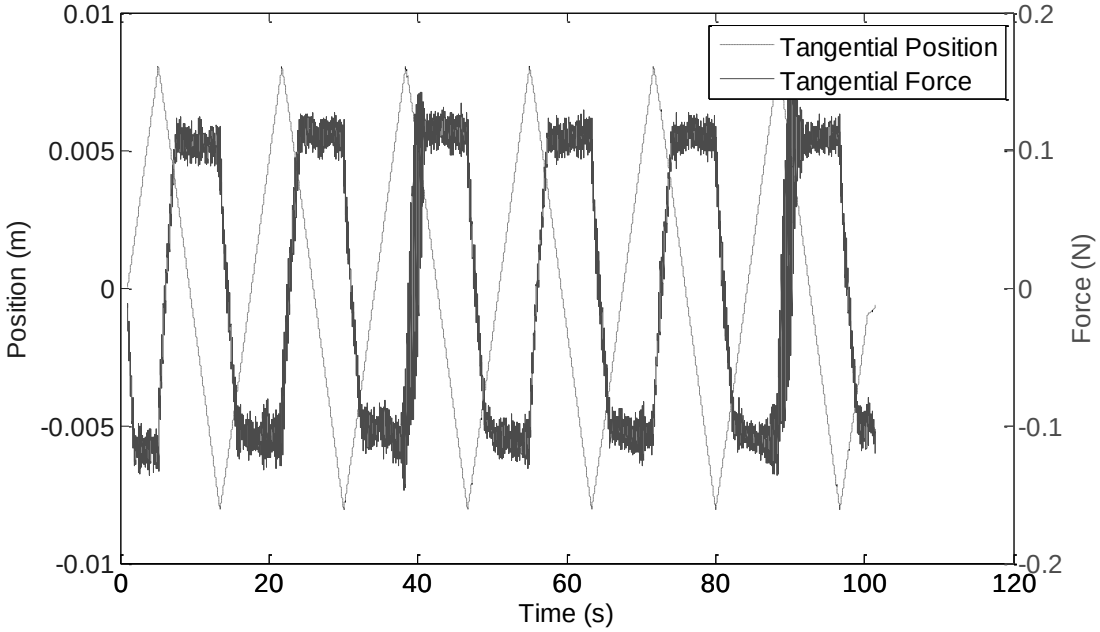

Figure 2.17 Friction test sample data

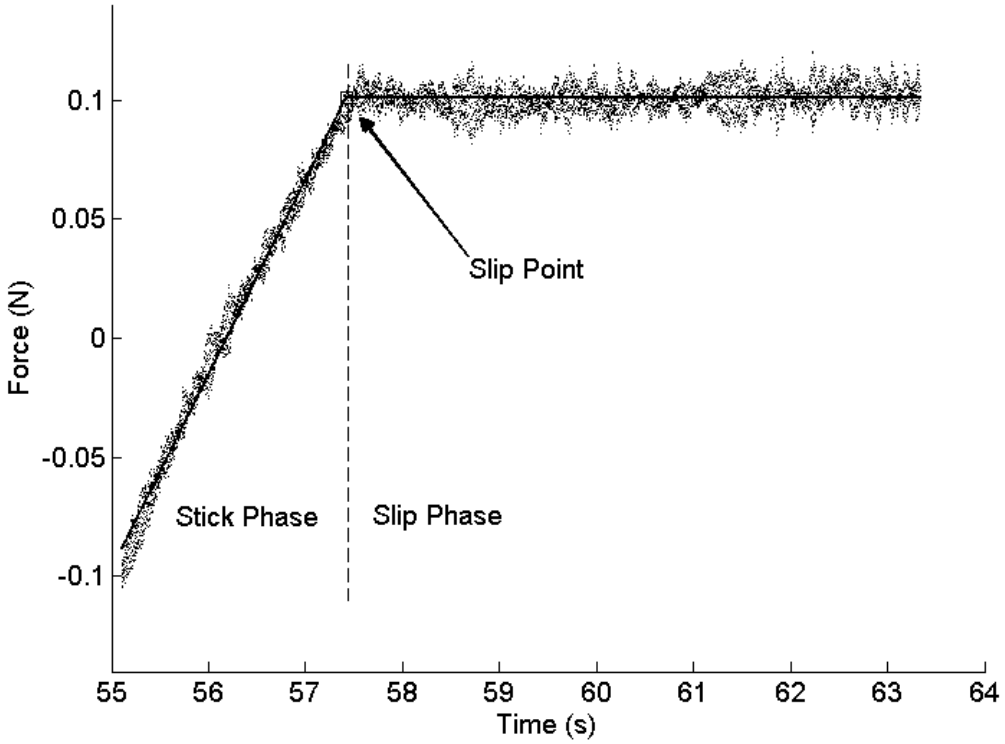

Figure 2.18 Friction test sample data zoomed in on single stroke

As the tactor moves and stretches the skin, tangential force builds until tactor slips.

and the second line constrained to be horizontal. The intersection of the two curve fitting lines, or equivalently, the value of the second horizontal line, was considered to be the slip point used to calculate the skin coefficient of friction. The coefficient of friction for all ten strokes was averaged and recorded, typically producing a standard deviation of less than 3% of the averaged coefficient of friction.

2.2.3.2 Dynamic Skin Stretch

Figure 2.19 shows a plot of the actual tangential force and actual position versus time from a typical dynamic skin stretch test. Note that position was intentionally offset from the force for better display clarity. The tactor was commanded to follow a desired sinusoidal trajectory for each of 26 frequencies from 5 rad/sec up to 150 rad/sec.

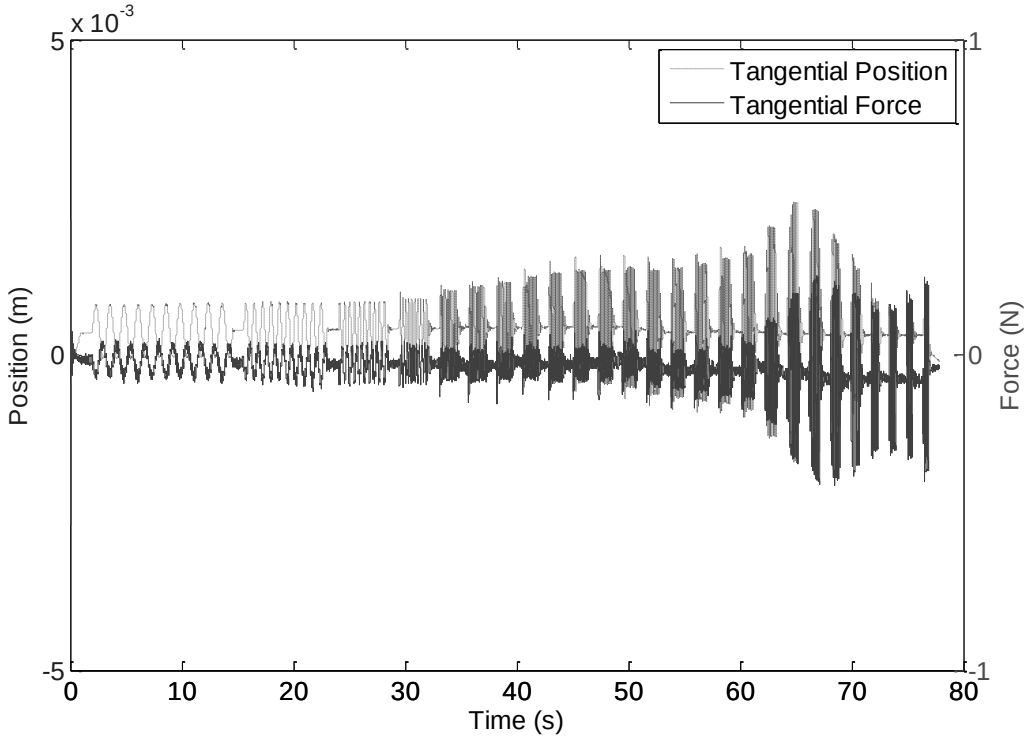

Figure 2.19 Dynamic skin stretch data: actual tangential position and force vs. time

Preliminary tests showed that the actual sinusoidal position trajectory of the tactor degraded at higher frequencies at or above about 150 rad/sec, likely due to the control algorithm used and the mass of the flotor and tactor. To isolate the dynamic properties of the skin from those of the MLHD flotor, actual position data was considered to be the input to the skin as long as it remained sinusoidal in form, and actual skin force response data was used as the output to isolate the human skin dynamics from the rest of the MLHD system.

Figure 2.20 shows the same data plot zoomed in on the section of the data with an input frequency of 5 rad/sec. The data appear as two sinusoidal waveforms offset by some phase shift for each of 26 different frequency inputs, with a one second delay

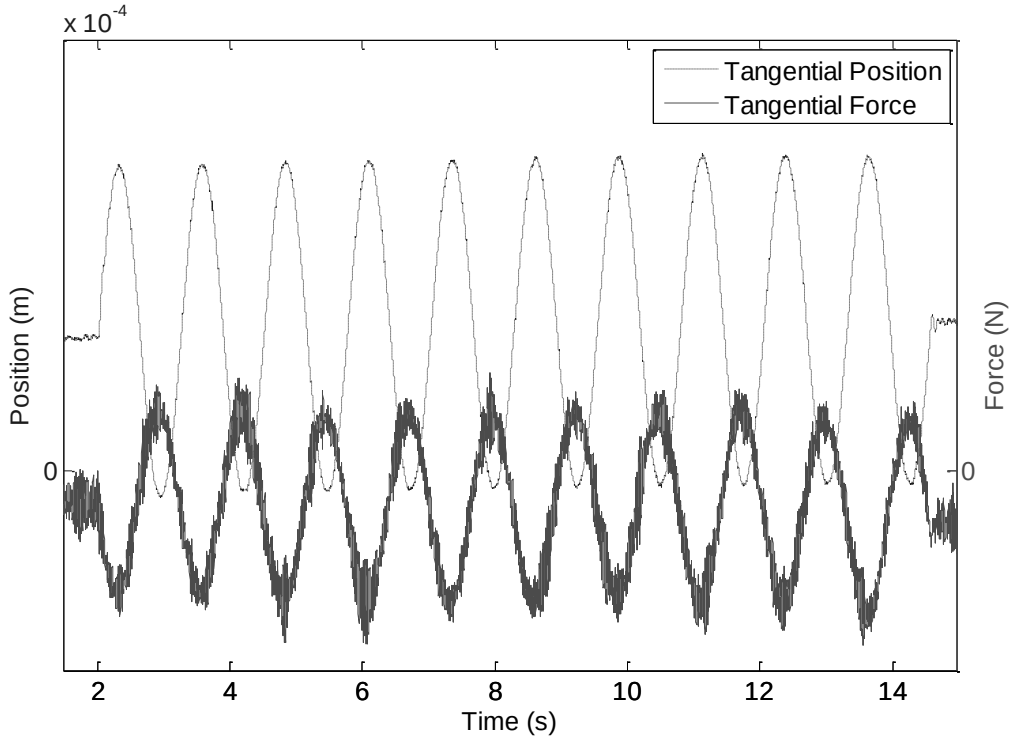

Figure 2.20 Dynamic skin stretch data zoomed in on one frequency: tangential position and force vs. time

separating each frequency input. The data was separated into groups by original frequency input, and each waveform was shifted vertically by its mean value to center it vertically about zero.

The discrete fast Fourier transform of each waveform was taken to determine each waveform's amplitude and frequency. The phase shift was calculated by finding the differences between the phases of the two waveforms, and the magnitude ratio was calculated by dividing the force amplitude by the position amplitude. The magnitude ratio and phase shift of each of the 26 frequencies was recorded, and the magnitude ratios and phase shifts were converted to Decibels and degrees, respectively. A Bode plot was created by plotting all 26 magnitude ratios and phase shifts versus actual position input frequency on a semi-log scale. An example Bode plot is shown in Figure 2.21. From the Bode plot, the DC gain and the peak amplitude ratio and frequency were determined and used to calculate the mass, damping, and stiffness constants of the skin test location in question.

2.2.3.3 Viscous Damping

Figure 2.22 shows a plot of the scaled tangential force and position data versus time from a typical viscous damping test using un-lathered shower gel. The tactor was commanded to follow a reciprocal stroking trajectory while doubling stroking velocity every two strokes. Likely due to the low maintained normal force and low measured tangential force, the data was much noisier than other test methods. Using the same piecewise regression algorithm as was used in the friction test analysis, the average

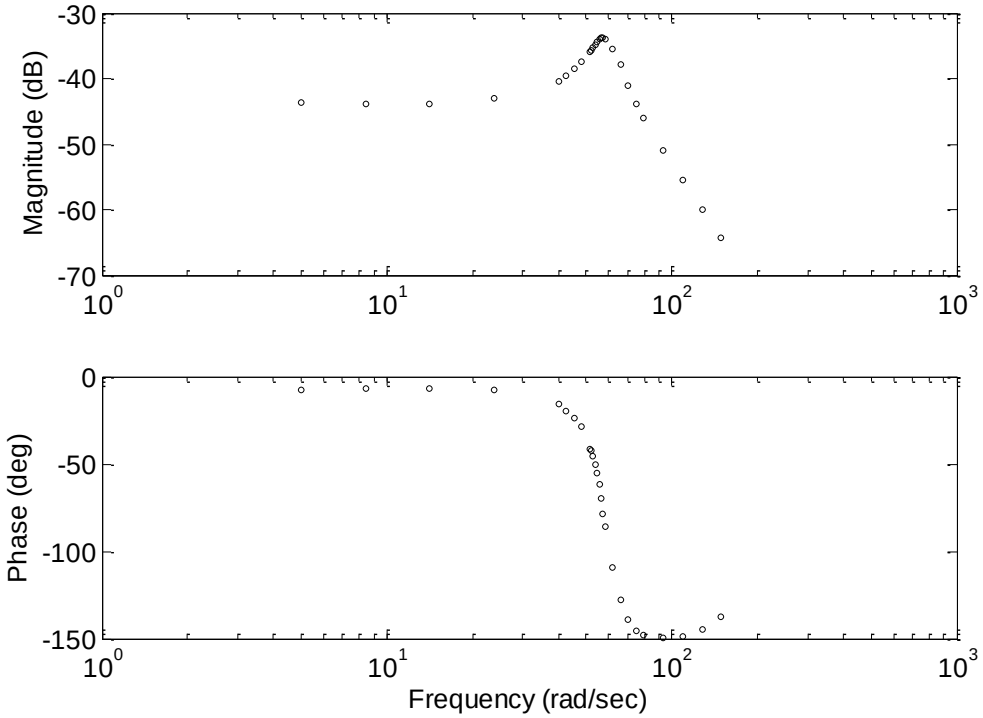

Figure 2.21 Sample Bode plot from dynamic skin stretch data

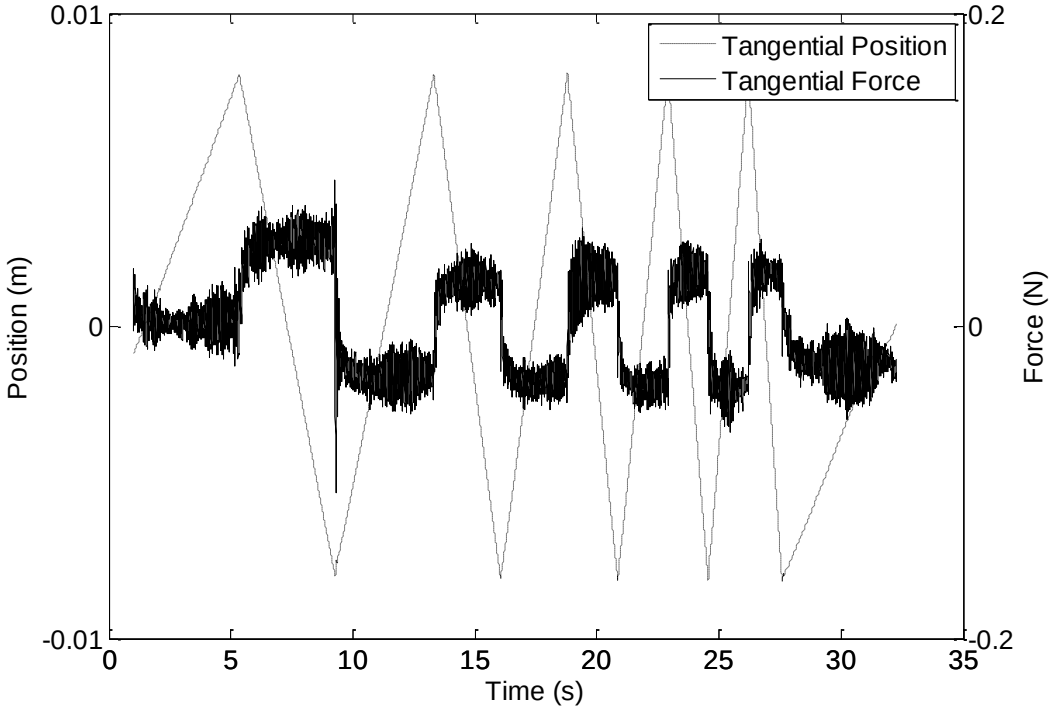

Figure 2.22 Sample viscous damping data: actual tangential position and force vs. time

tangential sliding force was determined for each stroking velocity. The tangential force was plotted versus sliding velocity for every test involving gel or foam. Figure 2.23 shows a sample of a force versus velocity plot.

A line was fit to the plotted points, and if the correlation coefficient was greater than or equal to 0.9, the slope of the fitted line was considered a good estimation of the viscous damping coefficient. If the correlation coefficient was lower than 0.9, the data were considered poor or too noisy to use, and was excluded.

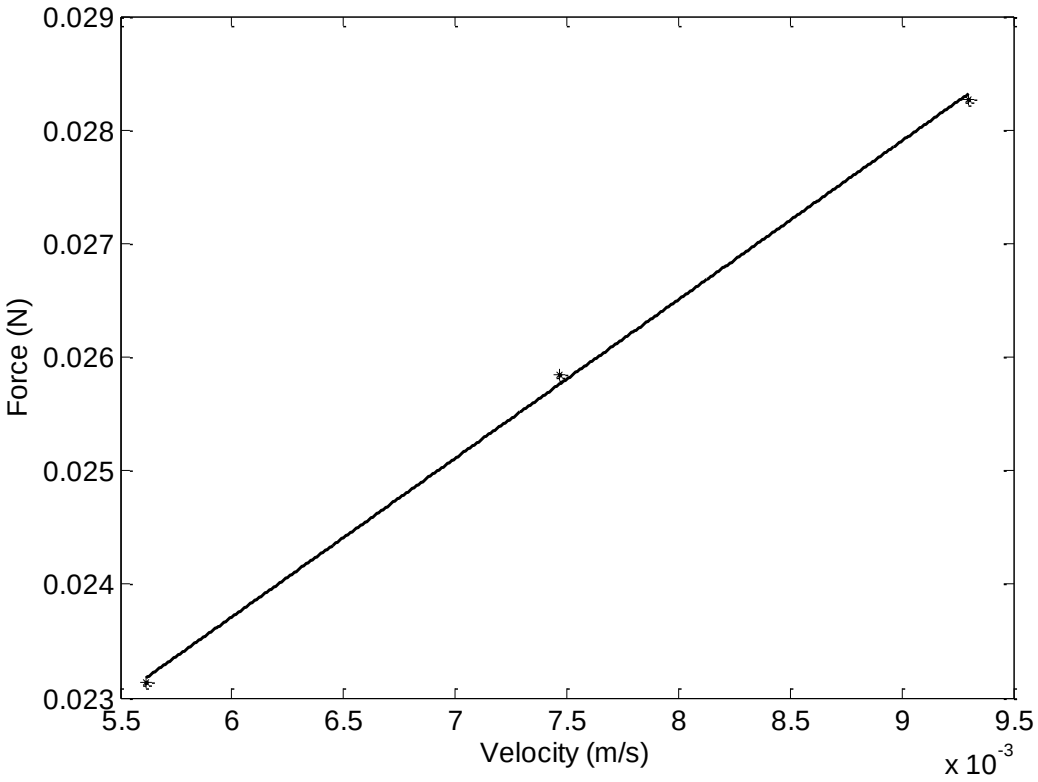

Figure 2.23 Sample plot of tangential force vs. velocity and linear fit.

In this case, the slope of the linear fit is considered to be the viscous damping constant (1.331) since the correlation coefficient (0.98) is greater than 0.9

2.2.4 Preliminary Experiments

To verify test repeatability, preliminary experiments were conducted nine times on one 28-year-old Caucasian male. Only four of the eight shower gels were used in the experiments, and shower gel ordering was pseudo-randomized among the four skin test locations. Skin testing was considered to be a form of destructive testing since skin condition changes with time, environment, and shower gel treatment. The experiments were conducted at least 24 hours apart to allow the skin time to return to steady-state conditions after each shower gel treatment. As detailed previously, friction and dynamic skin stretch tests were conducted on untreated skin and twice more after shower gel treatment: once immediately following treatment and rinsing of the skin, and again 20 minutes later. These three skin moisture states are referred to as Untreated, Rinsed, and Final, respectively. During shower gel application, the viscous damping test was conducted twice: once to characterize the interface of skin and gel, and again to measure the interface of skin and gel after being lathered into foam. These two viscous damping tests are referred to as Viscous Damping Gel, and Viscous Damping Foam, respectively.

Initially, it was hypothesized that the Rinsed and Final treated skin friction coefficients for each skin location needed to be normalized or offset by the respective initial Untreated skin friction in order to measure only the change in skin friction from shower gel treatment relative to the Untreated skin condition. However, the Rinsed and Final skin friction measurements indicated that the Untreated dry skin condition did not consistently affect the treated skin friction measurements, and also proved the equipment and testing procedures to be reasonably repeatable.

Figure 2.24 shows a bar plot of the mean skin friction coefficients before and after

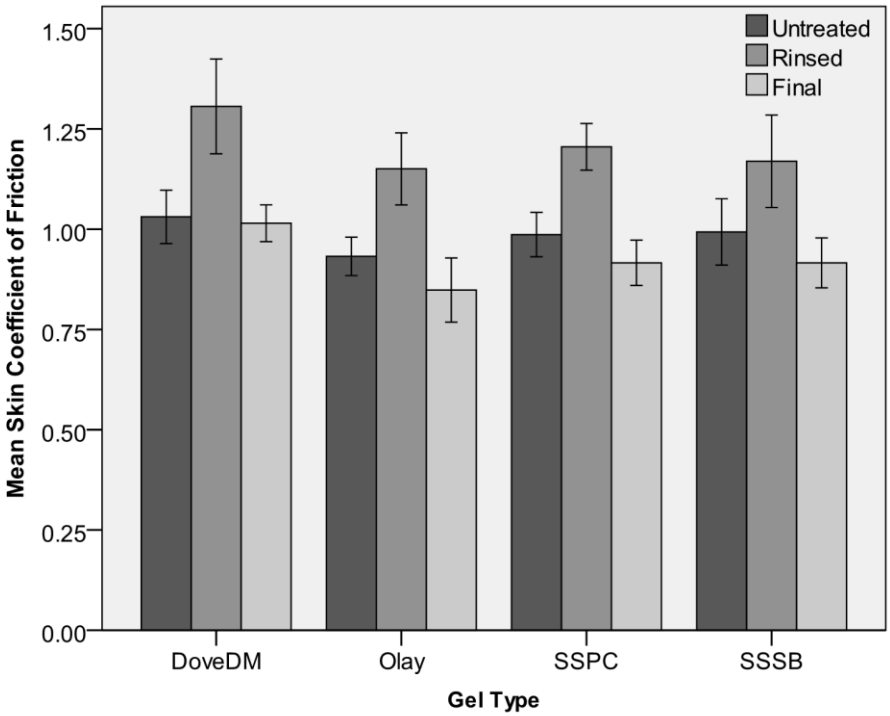

Figure 2.24 Mean skin coefficient of friction versus gel type from preliminary tests, with 95% CI error bars.

Higher skin coefficient of friction indicates higher skin hydration levels. This figure shows the skin coefficient of friction averaged from 9 tests conducted on one 28-year-old Caucasian male. The data shows a significant difference between Dove Deep Moisture and Olay Final skin friction coefficients. The figure also shows Final skin friction may be the most repeatable test and the most likely test to show differences between shower gel friction effects.

shower gel treatment with 95% confidence interval error bars. It is evident from the figure that Final skin friction may provide the best repeatability and thus the most significant difference between shower gel effects as well. Based on friction results of preliminary testing on a single test subject as shown in Figure 2.24, Dove Deep Moisture consistently yielded the highest skin friction out of the four gels tested for both Rinsed and Final skin moisture states. This result implies that of the four gels tested, Dove Deep Moisture maintained the greatest skin hydration for the 28-year-old male Caucasian. More importantly, although the skin of other test subjects may react somewhat differently

to different shower gels, the results show that skin friction is a repeatable test, and that specifically Final skin friction may be a better indirect skin hydration metric than Rinsed skin friction.

Figures 2.25 through 2.27 show the bar plots of mean skin mass, stiffness constant, and damping constant, respectively, as calculated from the dynamic skin stretch data measured before and after each shower gel treatment during preliminary testing. Figure 2.25 shows that skin mass generally increased with shower gel treatment, although only Soft Soap Shea Butter produced a Final skin mass significantly higher than the Untreated skin mass.

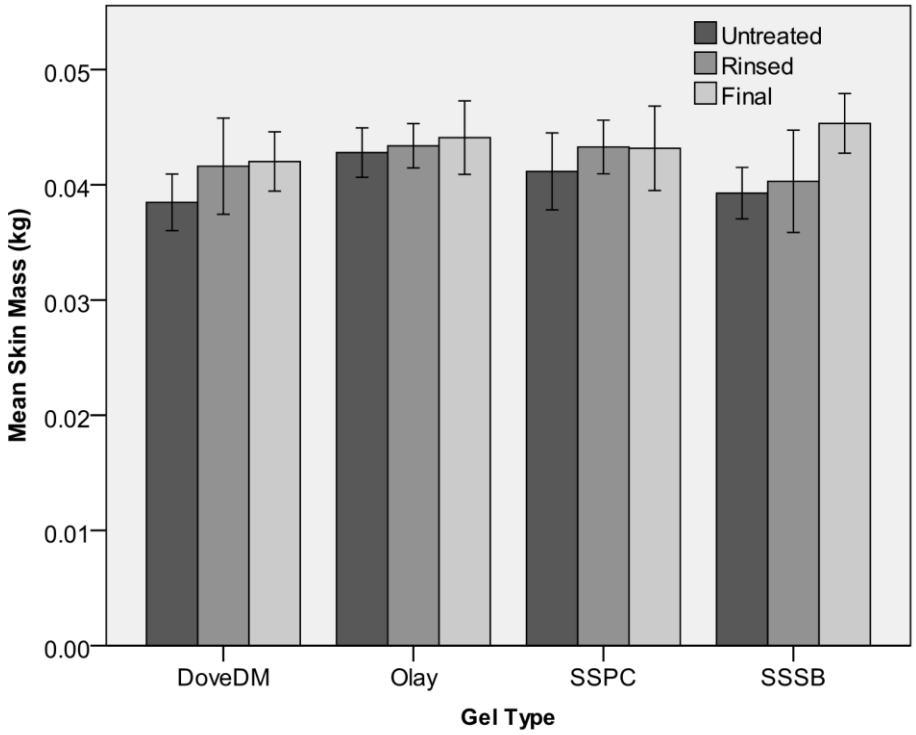

Figure 2.25 Mean skin mass vs. gel type from preliminary dynamic skin stretch tests, with 95% CI error bars.

This figure shows skin mass averaged from nine tests on one 28-year-old Caucasian male, with smaller error bars indicating better repeatability.

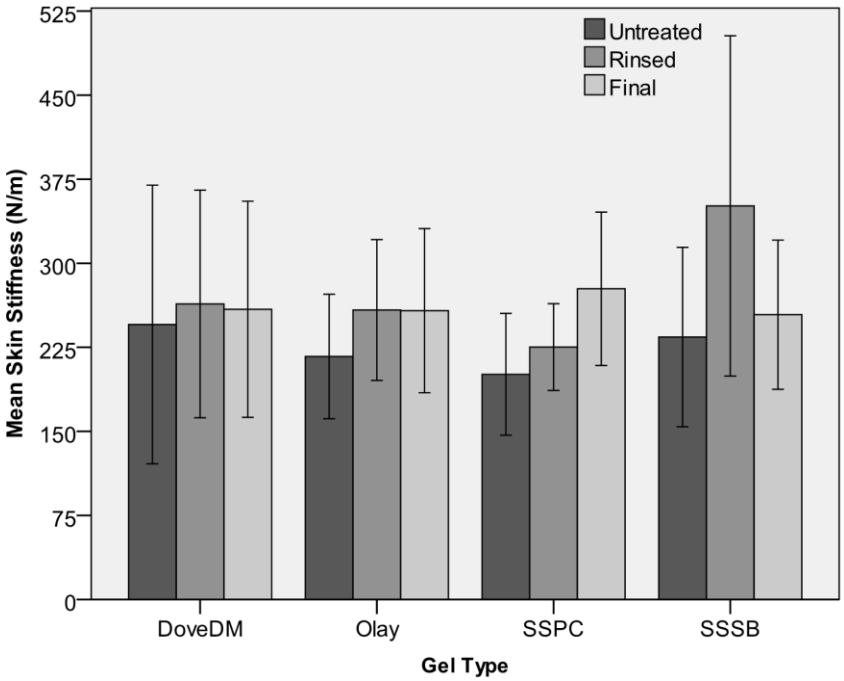

Figure 2.26 Mean skin stiffness constant vs. gel type from preliminary dynamic skin stretch tests, with 95% CI error bars.

This figure shows skin stiffness constant averaged from nine tests on one 28-year-old Caucasian male, with large error bars indicating poor repeatability.

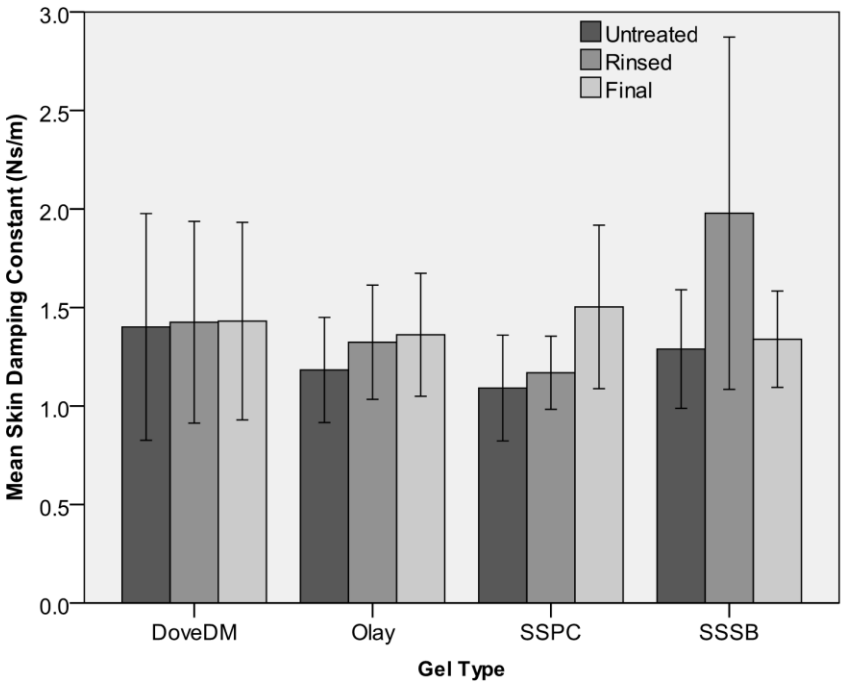

Figure 2.27 Mean skin damping constant vs. gel type from preliminary dynamic skin stretch tests, with 95% CI error bars

As shown by Figures 2.26 and 2.27, skin stiffness and damping constants had much larger 95% confidence interval error bars, showing the repeatability of these parameters to be too poor to allow any differentiation between moisture states or between gel effects. Although the mass, stiffness, and damping parameters derived from dynamic skin stretch data showed little or no difference between effects of different shower gels, it was decided that skin mass in particular may still yield significant results in human testing, and so the dynamic skin stretch testing would still be included in future testing.

Figure 2.28 shows the mean Viscous Damping Gel and Viscous Damping Foam coefficients used to characterize the skin and shower gel effects during treatment. The plot shows a major difference in gel type for Viscous Damping Gel, and very little difference between shower gels for Viscous Damping Foam. Since viscous damping is a measure of the amount of energy lost during contact and relative movement between a solid body and a liquid, Figure 2.28 indicates that Soft Soap Shea Butter gel may feel more thick or resistant to flow when in gel form on skin. When in lathered foam form on skin, however, no significant difference is seen in gel type. The viscous damping test thus appears to be generally repeatable, and most likely to show significant differences between different shower gels when used to measure the viscous damping of gel on skin.

Again, although it can be assumed that the skin condition of a single test subject may change somewhat over time, depending on factors such as humidity and test subject activity, preliminary test results showed that the MLHD setup produces reasonably repeatable skin friction and viscous damping measurements, and reasonably repeatable skin mass parameters as calculated from dynamic skin stretch data.

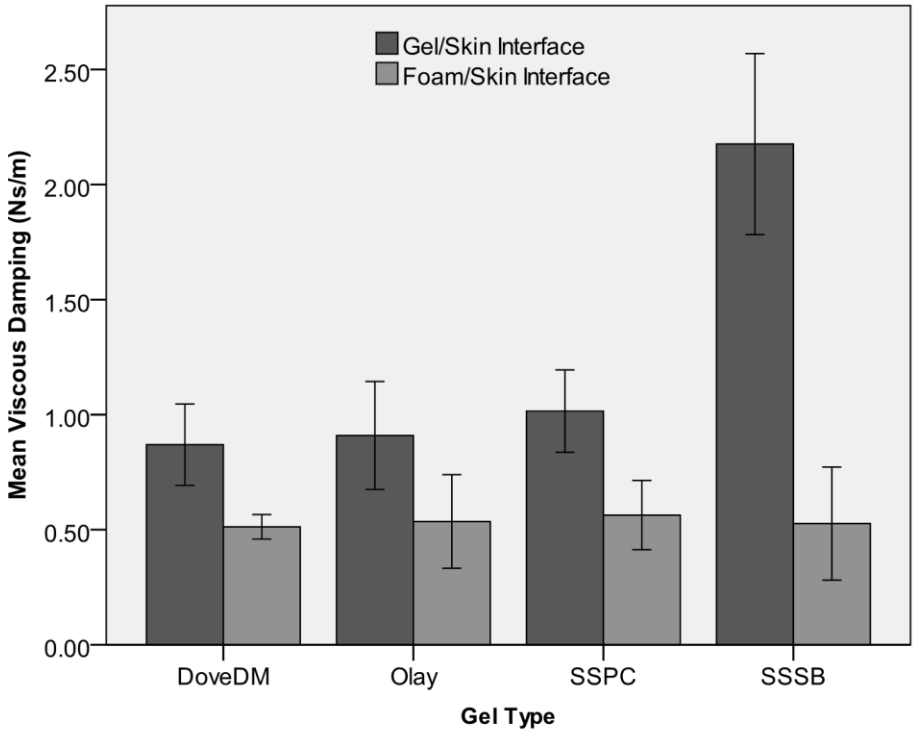

Figure 2.28 Mean viscous damping constants vs. gel type from preliminary tests, with 95% CI error bars

This figure shows viscous damping data averaged from 9 preliminary tests on one 28-year-old Caucasian male, with small error bars indicating good repeatability. Since significant differences are shown between gels, it is assumed that data averaged from future experiments on multiple subjects will yield strong significance between gels.

2.2.5 Final Experimental Design Considerations

To prevent potential effects from environmental effects during each test, the order of shower gels was pseudo-randomized among the four skin test locations and across two test sessions. Table 2.3 shows the shower gels listed and numbered consecutively in alphabetical order with corresponding abbreviations.

The Balanced Latin Squares method was used to create an eight by eight square ordering matrix, as shown in Table 2.4. The first four columns were used to indicate shower gel order for the first test session, and the remaining four columns determined

shower gel order for the second test session. The total number of test subjects was planned to be a multiple of eight to use every row of the ordering matrix equally.

Table 2.2 Shower gel numbers, names, and corresponding abbreviations

Shower Gel Numbers Key			
Number	**Brand**	**Name**	**Abbreviation**
1	Dial	Yogurt Aloe Vera	Dial
2	Dove	Deep Moisture	DoveDM
3	Dove	Go Fresh	DoveGF
4	Olay	Body Butter Ribbons	Olay
5	SoftSoap	Pure Cashmere	SSPC
6	SoftSoap	Nutri-Serums	SSNS
7	SoftSoap	Ultra Rich Shea Butter Crème	SSSB
8	Bath & Body Works	Sweet Pea Gel	SweetPea

Table 2.3 Balanced Latin Squares ordering

Balanced Latin Squares Ordering								
Test Subject	**Shower Gel Number**							
	Session A				**Session B**			
1, 9, 17, 25	1	2	8	3	7	4	6	5
2, 10, 18, 26	2	3	1	4	8	5	7	6
3, 11, 19, 27	3	4	2	5	1	6	8	7
4, 12, 20, 28	4	5	3	6	2	7	1	8
5, 13, 21, 29	5	6	4	7	3	8	2	1
6, 14, 22, 30	6	7	5	8	4	1	3	2
7, 15, 23, 31	7	8	6	1	5	2	4	3
8, 16, 24, 32	8	1	7	2	6	3	5	4
Test Location	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**

CHAPTER 3

EXPERIMENTAL RESULTS

3.1 Introduction of Results

A total of 32 human subjects were tested. GPower 3.1 software was used to conduct power analyses throughout human testing to estimate the total number of subjects required. Each subject participated in two test sessions in order to test all eight gels on each subject, with test sessions separated by at least three days.

Friction and dynamic skin stretch tests were performed three times on each test subject: once to measure the skin properties before shower gel treatment, again to measure the skin properties immediately after shower gel treatment, rinsing, and drying, and once more to measure the skin properties 20 minutes after rinsing. These three skin moisture states are referred to as Untreated, Rinsed, and Final, respectively. The viscous damping test was conducted twice: once to characterize the interface of un-lathered gel on skin, and again to measure the interface of lathered gel on skin. These two viscous damping tests are referred to as Viscous Damping Gel, and Viscous Damping Foam, respectively.

All levels of all dependent variables were found to have relatively normal distributions. Viscous damping was the only test with a few levels that yielded only

marginal normality, likely because a number of data points were omitted as bad data. Although additional viscous damping data would likely improve normality, the significance of the effect of gel type on viscous damping was found to be very strong [$p<0.001$]. It was thus assumed that additional data would very likely not change the significance significantly.

All statistical analyses were performed in SPSS and MATLAB. The primary null hypothesis for all experiments in this study is that there is no difference between the effects of any of the eight shower gels on human skin. To address the primary null hypothesis, the statistical analyses of all data focused on identifying between-subject correlations in data by shower gel type. A one-way ANOVA was used to determine effects of shower gel type on each dependent variable.

Additional controlled independent factors include shower gel test order and skin test location, which would address possible effects from environmental effects and differences in skin properties between skin test locations within test subjects. Uncontrolled independent factors were also identified from test subject age and gender. A secondary null hypothesis was that no other independent factors affect the significance of shower gel type on any of the dependent variables. To address the second null hypothesis, a two-way ANOVA was used to investigate the effects of shower gel type on each dependent variable while accounting for each of the other independent factors, one at a time. Figure 3.1 shows a histogram of test subject age and gender, as well as statistics for mean age and standard deviation within gender. Dependent variables were grouped into two categories: data collected from skin measurements before and after shower gel treatment, and data collected during shower gel treatment.

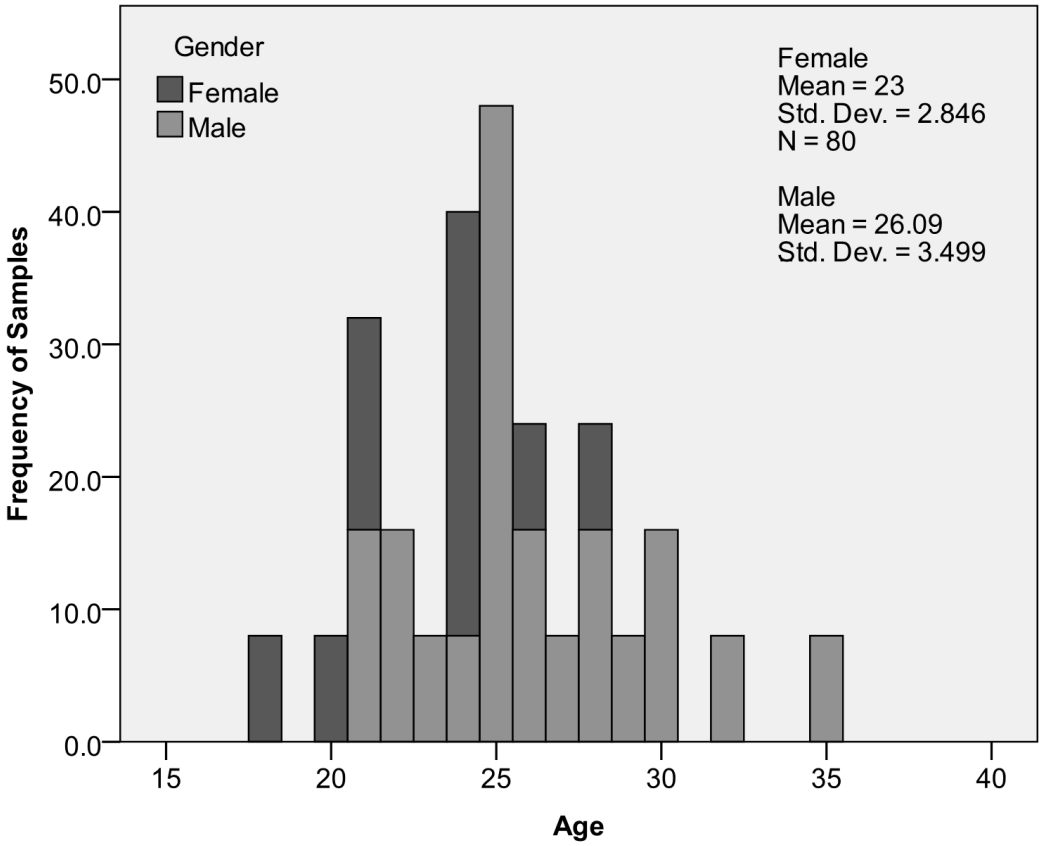

Figure 3.1 Histogram of subject age and gender

Total N = 256 samples from 32 test subjects and eight shower gels. Since age and gender were not controlled factors, some ages only contained subjects of one gender, and some ages contained only a single test subject or none at all.

In the first category, friction and dynamic skin stretch data were used to characterize skin properties for Untreated, Rinsed, and Final moisture states. In the second category, viscous damping measurements were used to characterize the interface of skin and un-lathered shower gel, and the interface of skin and shower gel after being lathered into foam. Note that it was found universally that skin test location had no significant effect on any dependent variables, thus effectively validating the initial assumption that there is no difference between skin test locations.

3.2 Friction Results

3.2.1 Final Skin Friction Results

Statistical analyses were performed first on friction data since preliminary experiments showed Final friction to be most repeatable, regardless of initial Untreated skin condition. A bar plot of the mean Final skin friction and 95% confidence interval for each shower gel as found from the data of all 32 test subjects is shown in Figure 3.2. The Final skin friction was not found to have statistically significant means between shower gels [$F(248,7)=1.94$, $p=0.063$]. However, when subject gender was taken into account, shower gel type was found to have a very significant effect on Final skin friction [$F(240,7)=2.855$, $p<.01$]. A Tukey post-hoc analysis showed the only significant difference in mean Final skin friction was between Olay and Sweet Pea gels, which yielded the highest and lowest mean Final skin frictions, respectively. Based on this result and the direct correlation between skin friction and skin hydration discussed in Chapter 1, it can be concluded that treatment with Olay gel yields significantly higher tested skin hydration than treatment with Sweet Pea gel.

A correlation study was performed to verify the preliminary testing assumption that there is no dependence of Final skin friction on initial skin condition. Contrary to preliminary test assumptions, however, a weak-to-moderate positive correlation was discovered between Initial and Final skin coefficients of friction (see Appendix A). To address this correlation, an additional variable was computed by dividing the Final skin friction by the Untreated skin friction value for each case. The resulting normalized Final coefficient of friction data is shown categorized by gel type with 95% confidence interval error bars in Figure 3.3. A Tukey post-hoc analysis revealed that Olay gel resulted in

normalized Final friction significantly higher than that of SweetPea gel [f(245,7)=2.845, p<.01]. It should be noted that this within-subjects normalization does not address the issue of between-subject variance due to differences in skin type or other uncontrolled factors between subjects. Additionally, since there is only one data point per subject for each shower gel and moisture state, an ANOVA analysis could not be used to take between-subject differences into account, and so another method was needed.

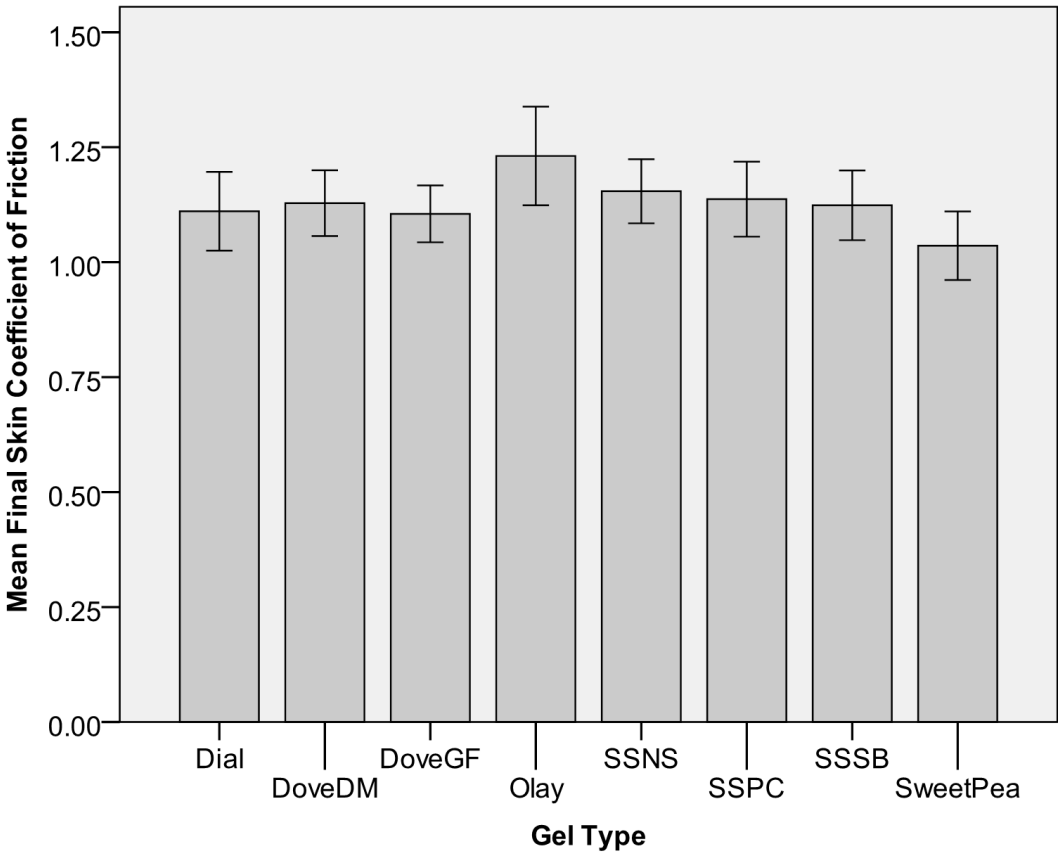

Figure 3.2 Mean Final skin coefficient of friction versus gel type, with 95% CI error bars

Higher skin coefficient of friction indicates higher skin hydration level. Without accounting for between-subject variation, no significant differences are shown between effects of shower gels on Final skin coefficient of friction.

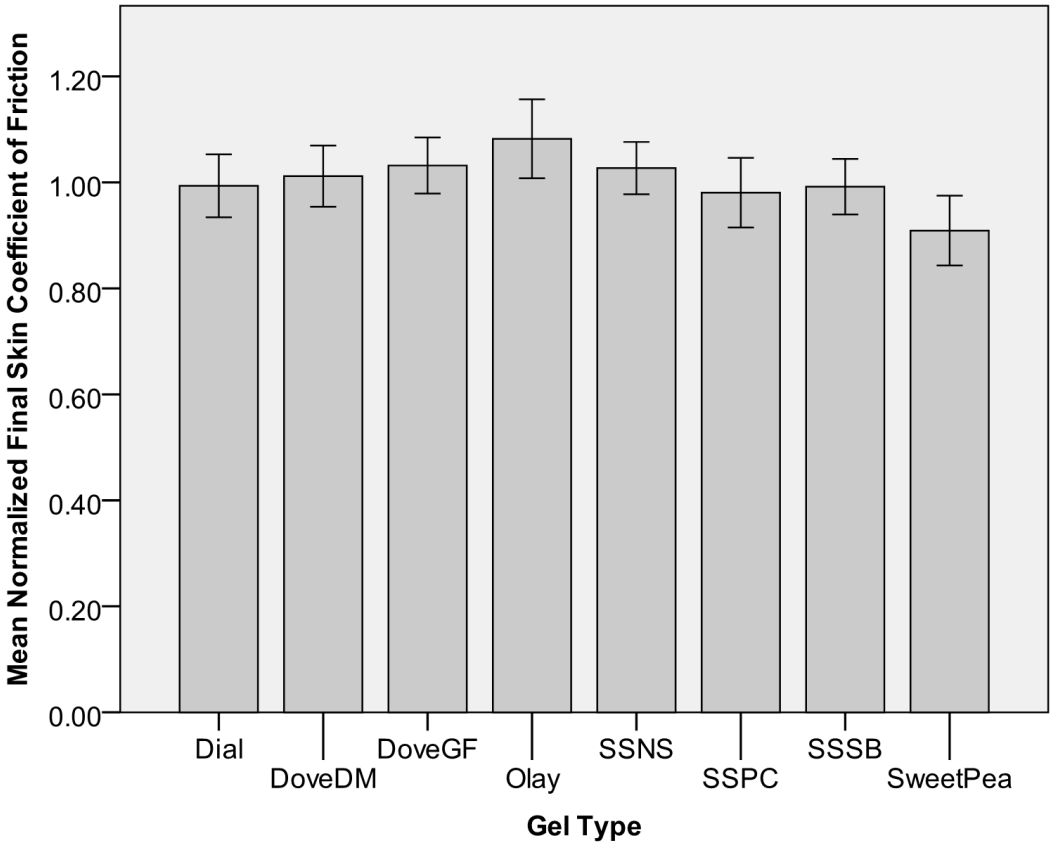

Figure 3.3 Mean normalized Final skin coefficient of friction versus gel type

The normalization of Final skin friction with respect to Untreated skin friction revealed a significant difference between the change in friction effects of Olay and SweetPea gels from before versus 20 minutes after treatment

3.2.1.1 Relative Comparison to Account for Between-Subject Variance

Although subject characteristics such as age or gender can be included to reveal the effects of shower gel type on absolute measures of skin friction, it was noted that only a relative comparison between shower gel effects on skin friction is needed to disprove the null hypothesis of this study. Additionally, no metrics were found to effectively normalize between subjects and thus account for between-subject variance due to differences in skin properties from subject to subject. To produce only a relative

comparison between shower gel effects without changing the distribution of each test subject's data, a method for shifting data was utilized, called Mean-Shifting [29].

First, each subject's mean friction coefficient was calculated from the eight friction coefficient measurements (one measurement per shower gel). Each subject's mean friction value was then subtracted from each of the subject's eight friction coefficients. Figure 3.4 gives a conceptual example of adjusting the friction data of two test subjects by shifting. Each data point represents friction measured from skin treated with one of the eight shower gels. Notice that although the data from both subjects may follow a similar trend by shower gel type, the data may exhibit a very different variance between subjects and within each subject's data. Thus the utility of mean-shifting is directly related to how well each subject's data follows the same relative trend by shower gel type, regardless of variance between subjects or within each subject's data.

Figure 3.5 shows the means and 95% confidence intervals of the Final friction data after shifting. It was found that shower gel type had a strong effect on the mean-shifted Final friction data [$F(248,7)=5.021$, $p<0.001$]. When taking into account age, gender, and shower gel test order, significance by shower gel type was maintained [$p<0.005$], but no two-way factor interactions were found to be significant. A Tukey post-hoc analysis estimated that a number of shower gels yielded significantly different effects on mean shifted Final skin friction. Olay gel produced a mean shifted Final friction significantly higher than Dial, Dove Go Fresh, Soft Soap Shea Butter, and Sweet Pea gels [$p<0.05$]. Again, since higher skin friction implies higher skin hydration level, it can be concluded that Olay gel maintains skin hydration better than Dial, Dove Go Fresh, Soft Soap Shea Butter, and Sweet Pea gels.

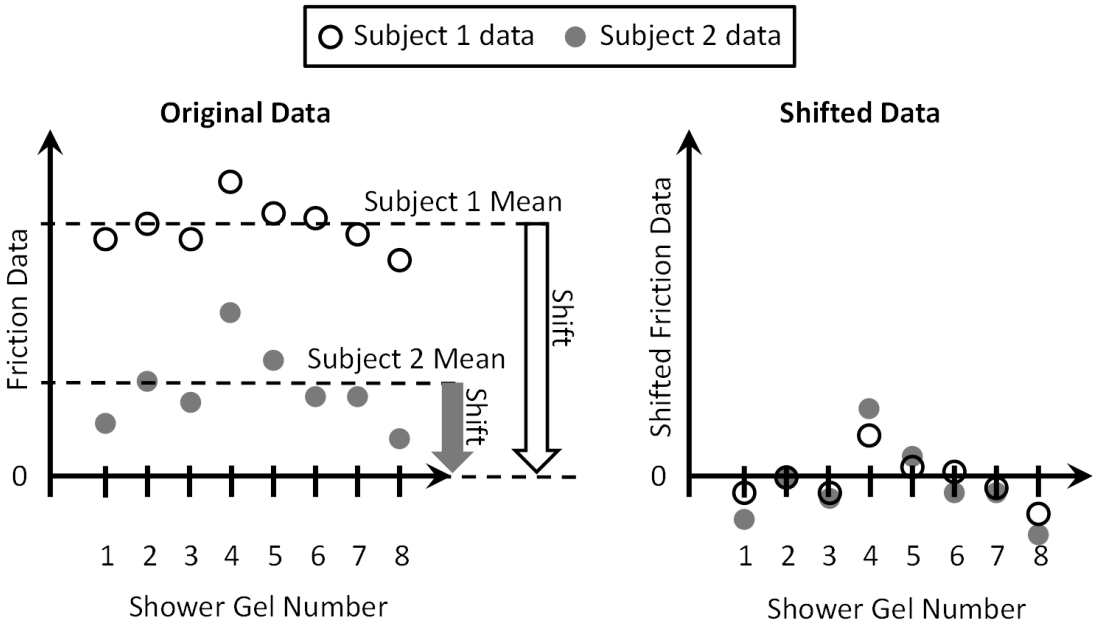

Figure 3.4 Conceptual example of shifting the data means to account for between-subject variance

The mean of each test subject's eight skin friction measurements (one from each shower gel treatment) is found, then subtracted from each of that subject's eight data points, thus effectively shifting each test subject's data to be centered at zero. This provides a relative comparison between shower gels instead of an absolute measurement of skin friction.

Sweet Pea gel produced a mean Final friction significantly lower than Olay and Soft Soap Nutri-Serums [$p<0.05$], meaning that Sweet Pea gel does not maintain skin moisture as well as Olay and Soft Soap Nutri-Serums. The Final friction means of Dove Deep Moisture and Soft Soap Pure Cashmere shower gels were found to be not significantly different from any of the other gels.

The normalized Final friction data was also mean-shifted to account for between-subject variance, thus providing data that accounts for both within-subject and between-subject variance. Figure 3.6 shows a bar plot of mean-shifted normalized Final friction

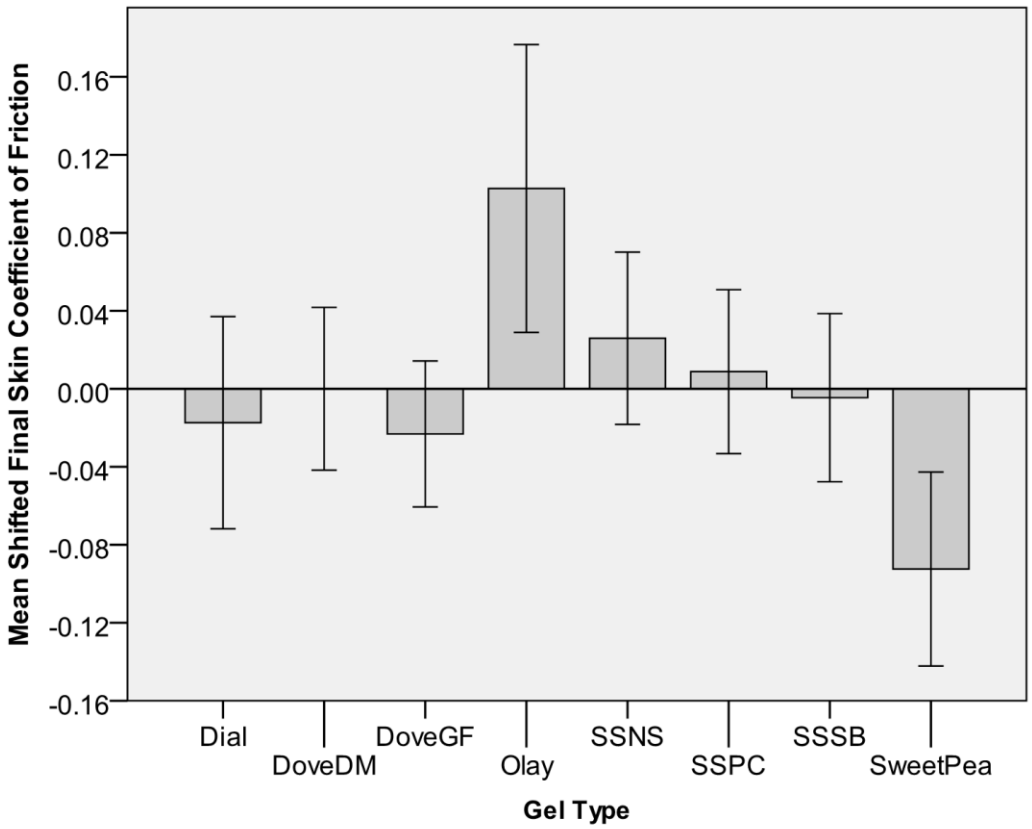

Figure 3.5 Mean shifted Final skin coefficient of friction versus gel type, with 95% CI error bars

After accounting for between-subject variance by shifting data means, a number of significant differences between shower gel Final friction effects are shown.

data versus gel type, with 95% confidence interval error bars. An ANOVA and Tukey post-hoc analysis showed the friction effects of SweetPea gel to be significantly lower than Olay, Dove Go Fresh, and SoftSoap Nutri-Serums [$(245,7)=4.016$, $p < 0.05$].

3.2.2 Rinsed Skin Friction Results

The mean coefficients of friction of Rinsed skin were found to be significantly different by shower gel type [$F(248,7)=3.172$, $p<0.005$]. Since the Rinsed skin results were already significant even with between-subject variance, no mean shifting was

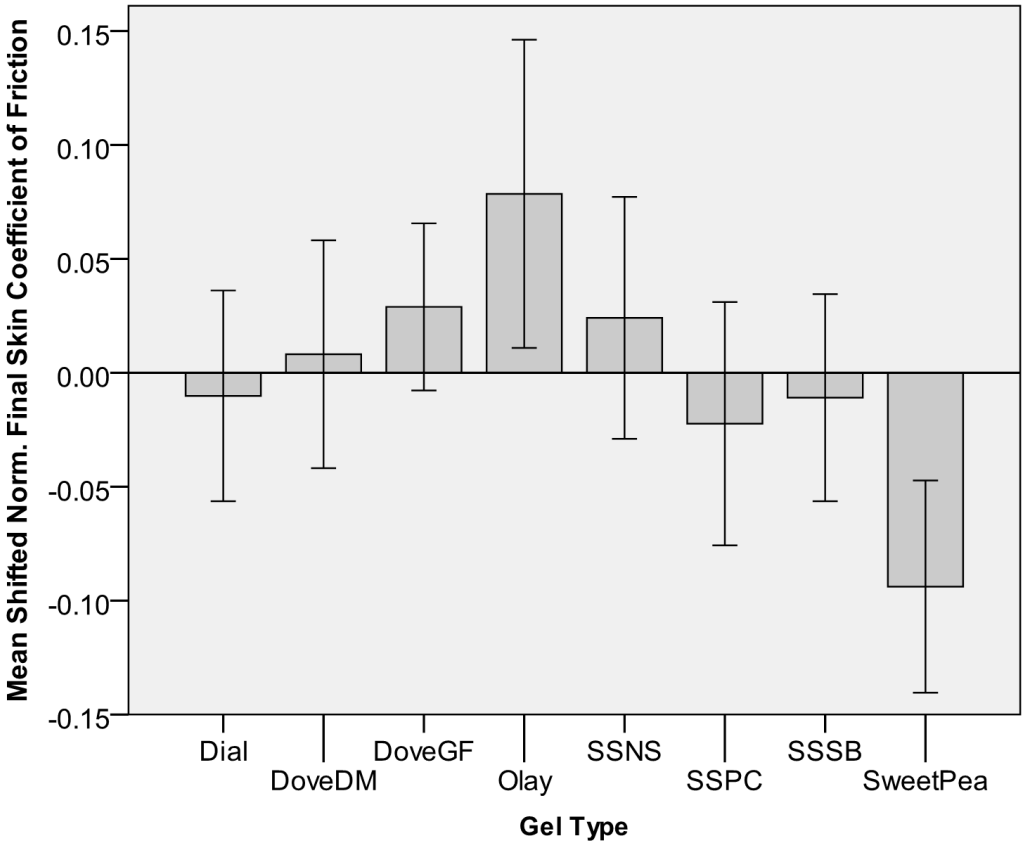

Figure 3.6 Mean-shifted normalized Final skin coefficient of friction

performed. Figure 3.7 shows a bar plot of the mean Rinsed friction coefficients for each shower gel. A Tukey post-hoc analysis showed that the effect of Sweet Pea gel on Rinsed skin friction was significantly lower than Rinsed friction effects of Dove Deep Moisture, SS Nutri-Serums, and SS Shea Butter gels [$p<0.01$]. This means that just as with Final skin friction, Sweet Pea Gel was found to maintain skin hydration the least out of any other gel. Besides significant differences from the effects of Sweet Pea Gel, no other gels showed effects that were significantly different from any other.

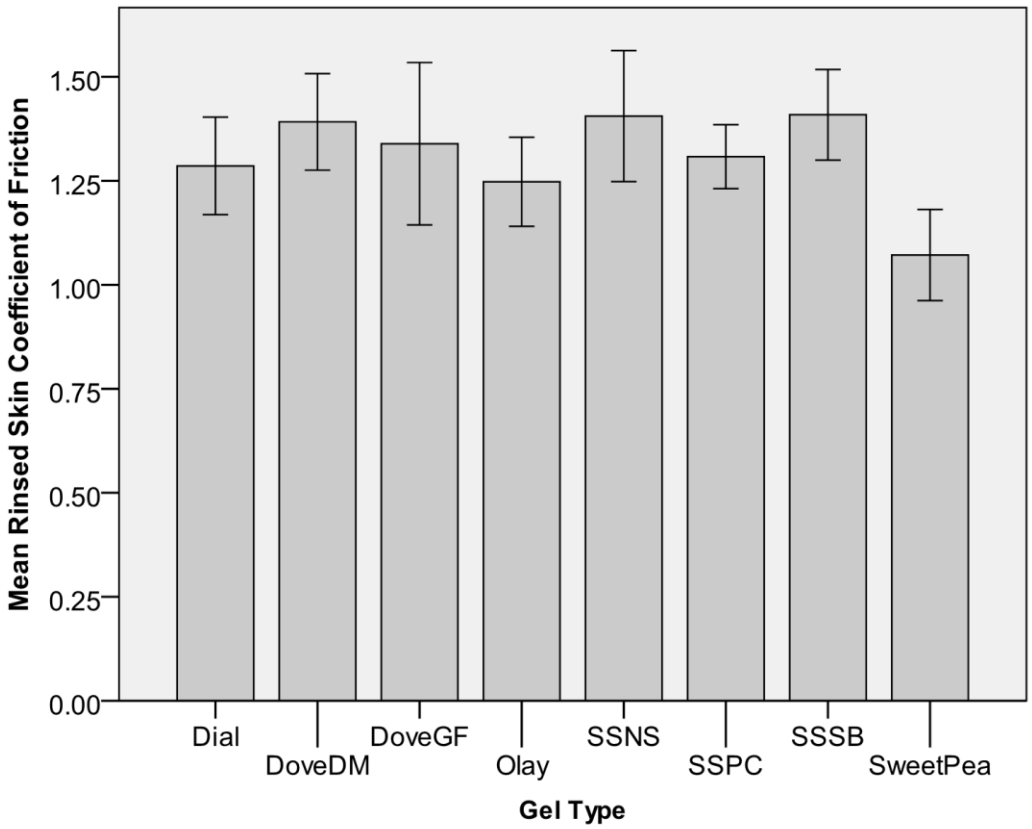

Figure 3.7 Mean Rinsed skin coefficient of friction versus gel type, with 95% CI error bars

Although Rinsed skin friction error is generally larger than Final skin friction error, significant differences are shown without need to compensate for between-subject variance.

3.2.3 Untreated Skin Friction Results

The effect of shower gel type on Untreated skin friction was not investigated since the skin had not yet been treated with any shower gels. It was found, however, that age and gender each had a significant effect on Untreated skin friction [age, $F(242,13)=10.808$, $p<0.001$] [gender, $F(248,7)=3.197$, $p<.005$]. Although ANOVA results indicate factor significance, it should be noted that uncontrolled demographic-based factors such as age and gender may not be sufficiently distributed throughout all other factors without testing

additional test subjects and controlling independent factor distribution. Untreated skin coefficient of friction was shown to be significantly higher for female subjects than for males, as shown in Figure 3.8. Figure 3.9 shows that there was no difference between the effects skin test location on Untreated skin friction.

3.3 Dynamic Skin Stretch Results

Dynamic skin stretch data consisted of three parameters as determined by system identification techniques to describe the properties of human skin: mass *m*, stiffness constant *k*, and damping constant *b*. Figures 3.10 through 3.12 show the means and 95% confidence interval error bars for the mass, stiffness, and damping parameters of all three

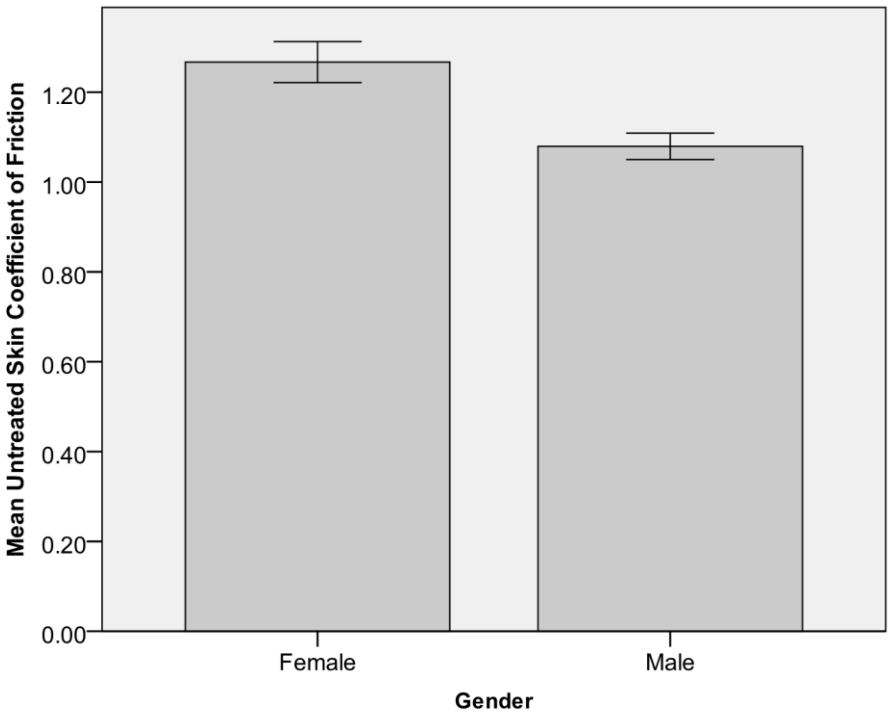

Figure 3.8 Mean Untreated skin coefficient of friction versus gender

Untreated skin friction results show that the skin of female subjects yielded a higher coefficient of friction than that of male subjects prior to shower gel treatment

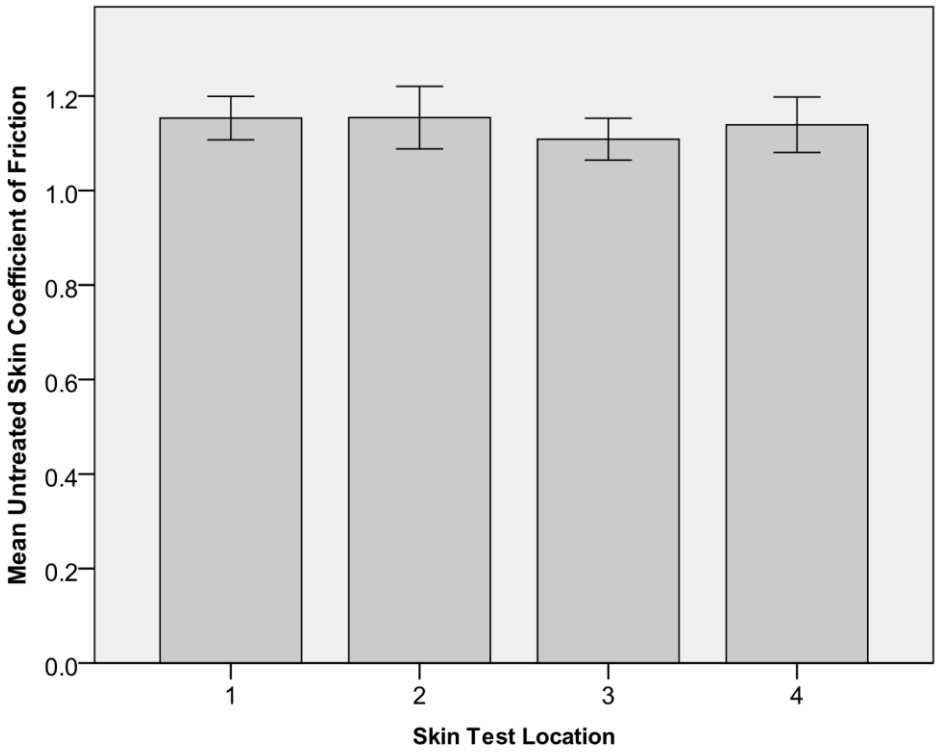

Figure 3.9 Mean Untreated skin coefficient of friction versus skin test location

Untreated skin coefficient of friction showed no significant differences between skin test locations, thus verifying that skin test location would likely not influence post treatment friction effects

skin moisture states versus shower gel type. None of the dynamic skin stretch parameters was found to be significantly affected by shower gel type only for either Rinsed or Final skin moisture states. When age was taken into account, however, shower gel type was found to have a significant effect on both the Final skin mass and Final skin damping constant, as shown in Table 3.1. There was also an interaction effect observed between shower gel type and age for both Final skin mass and Final skin damping constants. Similarly, when taking age into account, shower gel type was found to have a significant effect on Rinsed skin mass and Rinsed skin damping as shown in Table 3.2. Since human

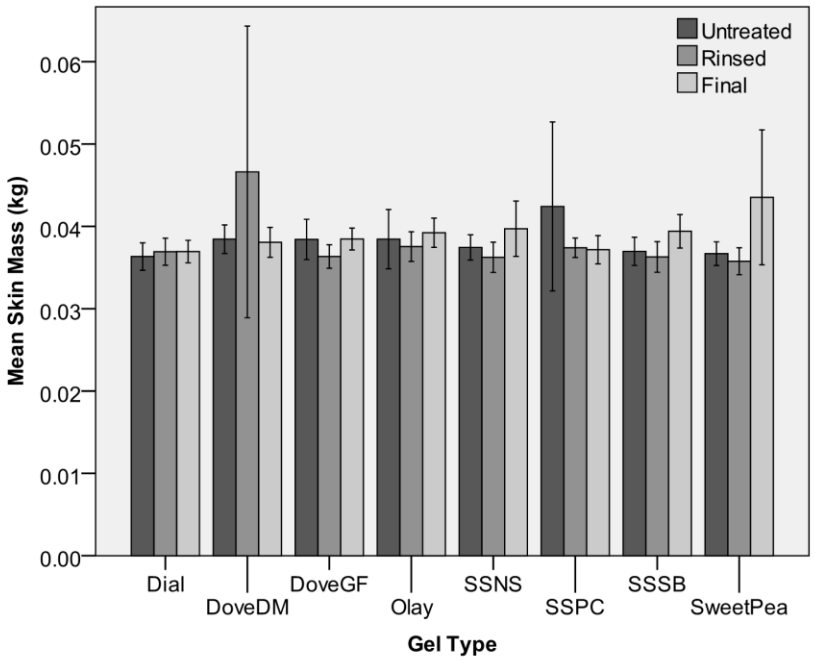

Figure 3.10 Mean skin mass vs. gel type from dynamic skin stretch data

Although error was generally low, there was no consistent difference observed between effects of shower gel type on skin mass from dynamic skin stretch data.

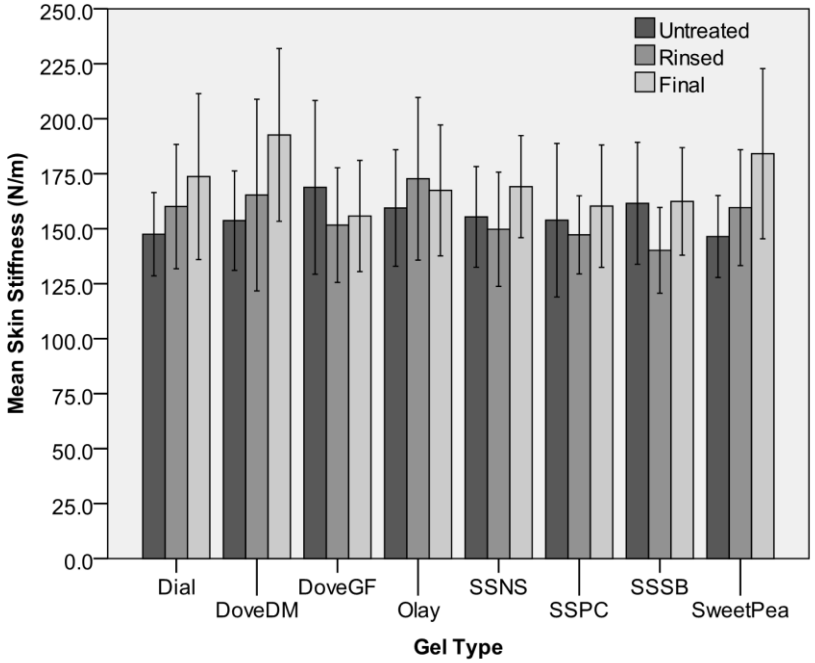

Figure 3.11 Mean skin stiffness vs. gel type from dynamic skin stretch data

No significant differences were found between moisture state or effects of gel type on skin stiffness

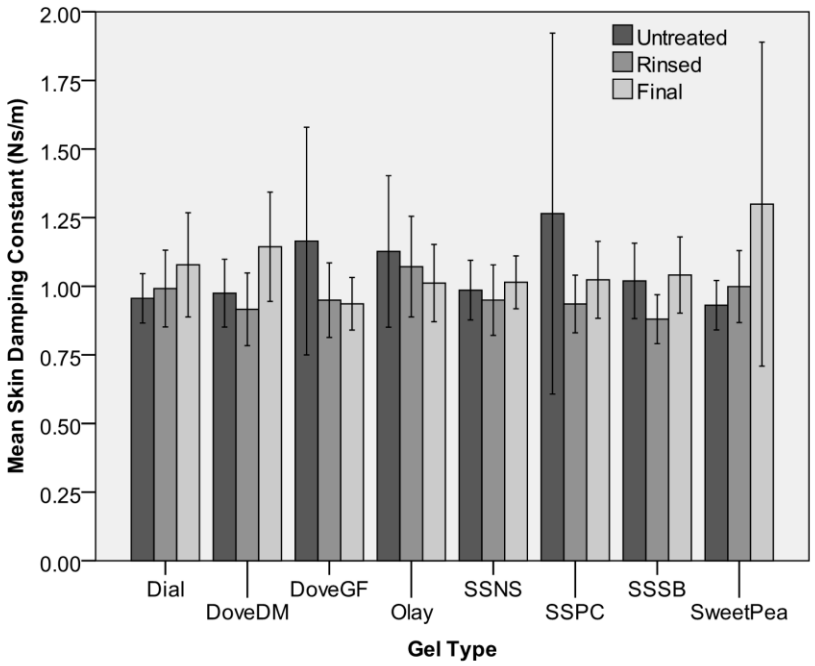

Figure 3.12 Mean skin damping constant vs. gel type from dynamic skin stretch data, with 95% CI error bars

No significant differences were found between moisture state or between the effects of gel type on skin damping

Table 3.1 Significant effects of shower gel type on Final mass, stiffness, and damping parameters (m, k, and b, respectively), α=0.05

	Final Skin Properties		
	m	k	b
Accounting for Age	[F(144,7)=8.464, p<.001]	none	[F(144,7)=7.667, p<.001]
Gel*Age	[F(144,91)=4.809, p<.001]	none	[F(144,91)=5.846, p<.001]

Table 3.2 Significant effects of shower gel type on Rinsed mass, stiffness, and damping parameters (m, k, and b, respectively), α=0.05

	Rinsed Skin Properties		
	m	k	b
Accounting for Age	[F(143,7)=63.933, p<.001]	[F(143,7)=2.100, p<.05]	none
Gel*Age	[F(143,91)=32.698, p<.001]	[F(143,91)=7595.296, p<.01]	none

skin appearance and other properties can be observed to vary significantly over time and between subjects, it is reasonable to assume that age or other demographic factors may strongly affect the dynamic parameters of human skin.

3.4 Viscous Damping Results

Shower gel type was found to have a strong effect on both the mean Viscous Damping Gel [$F(210,7)=22.723$, $p<.001$] and mean Viscous Damping Foam [$F(135,7)=4.084$, $p<.001$]. Figure 3.13 shows the means and 95% confidence interval error bars of the Viscous Damping Gel and Foam data. A higher viscous damping constant indicates a thicker gel or foam, or that more energy is required to lather or spread the gel or foam. Accordingly, the plots show that Soft Soap Nutri-Serums and Soft Soap Shea Butter are thicker or require more energy to lather on skin than any other gel when in gel form. Additionally, the plots imply that Sweet Pea gel lathered foam is thicker than most other shower gel foam.

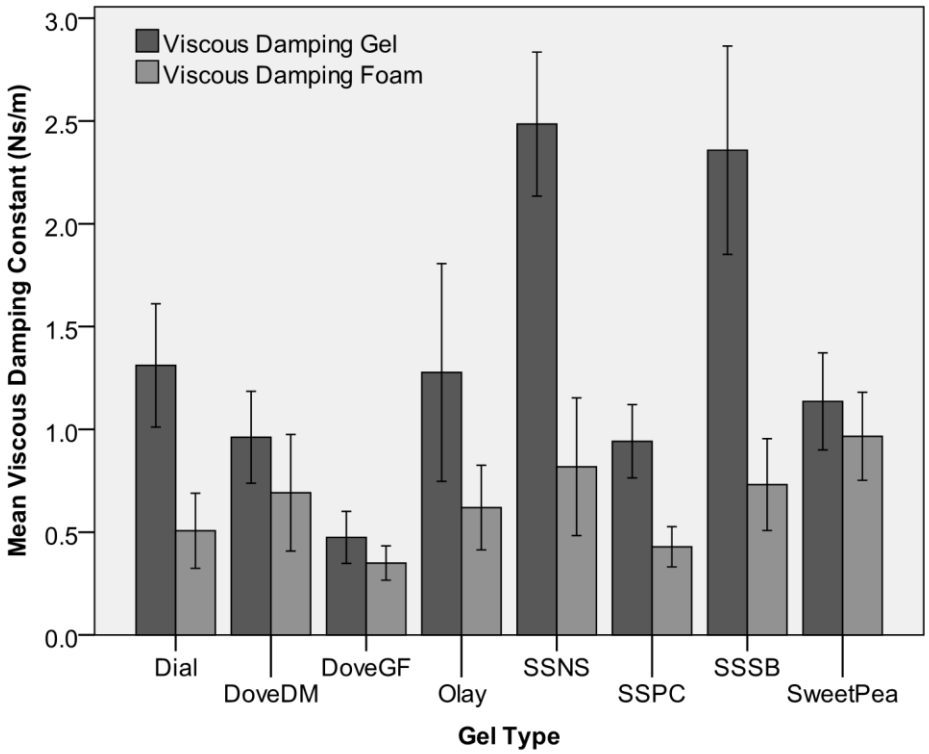

Figure 3.13 Mean viscous damping of gel and foam on skin, shown with 95% confidence interval error bars

The figure shows that shower gel type has a strong effect on the viscous damping constant of the interfaces between gel and skin and between foam and skin. Based on these results, Soft Soap Nutri-Serums and Soft Soap Shea Butter would likely require the most energy to lather, whereas Dove Go Fresh would require the least energy to spread on the skin or lather.

CHAPTER 4

CONCLUSION

4.1 Overview of Contributions

The haptic properties of human skin in response to treatment with each of eight specific shower gels were successfully characterized. A hybrid force/position control algorithm was developed to control a Magnetic Levitation Haptic Device to exert specific force and position trajectories on the skin of the human volar forearm in vivo. Three types of experiments were developed to measure the haptic properties of human skin and its interaction with shower gels before, during, and after shower gel treatments. Friction was used as the primary metric for measuring skin properties before and after shower gel application. Mass, stiffness constant and damping constant parameters as calculated from Dynamic Skin Stretch data were used as secondary metrics. Viscous Damping was used to characterize the interface between shower gel and human skin before and after lathering the gel into foam during shower gel treatments.

Preliminary experiments on a single test subject showed reasonable repeatability of these haptic tests. All three types of tests were used to test each of the eight shower gels on a total of 32 human subjects. Statistical analyses focused primarily on finding main effects of shower gel type, but also identified significant effects of uncontrolled demographic information of age and gender.

Mean Final friction results (20 minutes after treatment, rinsing, and drying) showed that Olay and Soft Soap Nutri-Serums consistently yielded the highest skin hydration, while Sweet Pea Gel consistently yielded the lowest. Various shower gels in between these extremes were not shown to have significantly-different effects on skin friction (and thus skin hydration). Dynamic skin stretch results were inconclusive except when test subject demographics were taken into account. Viscous damping results showed that SoftSoap Nutri-Serums and SoftSoap Shea Butter gels had the highest viscous damping constants and thus were likely to feel the thickest when applied to skin as un-lathered gels, while Dove Go Fresh, Soft Soap Pure Cashmere, and Sweet Pea gels yielded the lowest. Conversely, SweetPea gel was shown to produce the thickest foam on skin, while Dove Go Fresh still yielded the lowest viscous damping coefficient and thus the thinnest foam.

4.2 Future Work

Future work could focus on improving existing metrics (or finding additional metrics) for quantifying small changes in the haptic properties of human skin. Specifically, in place of current dynamic skin stretch testing, dynamic indentation testing could produce more useful and repeatable characterization of the mechanical properties of human skin. With indentation experiments, tactor slip would not be an issue as it may be in current dynamic skin stretch trajectories which are tangential to the skin surface. Additionally, as shown in related literature, quasi-static indentation could be used to measure the adhesive properties of skin.

A more advanced control algorithm could yield more conclusive results and better repeatability of the dynamic skin stretch test, as well as enable indentation testing. In the current control algorithm, the position/force selection matrix must be modified in real time in order to switch control modes in any degree of freedom. However, switching control modes typically introduces instability in the current controller. Thus a more advanced control algorithm could allow indentation tests, improve all test efficiencies and decrease total test time or decrease total number of subjects required.

Lastly, future experiments could be planned to better control distribution of test subject demographics or increase the total number of test subjects, especially for viscous damping tests. Such changes would enable more in-depth investigation into the effects of demographic information such as age, gender, climate, and ethnicity on skin properties. Additionally, a greater number of test subjects could improve gel type significance and reveal additional significant differences between shower gel effects.

APPENDIX A

STATISTICAL ANALYSIS TABLES

Correlations – All Test Subjects

Control Variables			Untreated Friction	Rinsed Friction	Final Friction
Subj Num & Test Order & Test Loc	Untreated Friction	Correlation	1.000	.281	.469
		Significance (2-tailed)	-	.000	.000
		df	0	248	248
	Rinsed Friction	Correlation	.281	1.000	.431
		Significance (2-tailed)	.000	-	.000
		df	248	0	248
	Final Friction	Correlation	.469	.431	1.000
		Significance (2-tailed)	.000	.000	-
		df	248	248	0

Tests of Between-Subjects Effects

Dependent Variable: Untreated Friction

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	.172[a]	7	.025	.514	.824
Intercept	328.038	1	328.038	6869.256	.000
Gel	.172	7	.025	.514	.824
Error	11.700	245	.048		
Total	340.037	253			
Corrected Total	11.872	252			

a. R Squared = .014 (Adjusted R Squared = -.014)

Tests of Between-Subjects Effects

Dependent Variable: Rinsed Friction

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	2.807[a]	7	.401	3.172	.003
Intercept	437.579	1	437.579	3461.819	.000
Gel	2.807	7	.401	3.172	.003
Error	31.348	248	.126		
Total	471.733	256			
Corrected Total	34.154	255			

a. R Squared = .082 (Adjusted R Squared = .056)

Tests of Between-Subjects Effects

Dependent Variable: Mean-Shifted Rinsed Friction

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	2.807[a]	7	.401	5.457	.000
Intercept	0.000	1	0.000	0.000	1.000
Gel	2.807	7	.401	5.457	.000
Error	18.222	248	.073		
Total	21.028	256			
Corrected Total	21.028	255			

a. R Squared = .133 (Adjusted R Squared = .109)

Tests of Between-Subjects Effects

Dependent Variable: Final Friction

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	.663[a]	7	.095	1.945	.063
Intercept	325.888	1	325.888	6695.033	.000
Gel	.663	7	.095	1.945	.063
Error	12.072	248	.049		
Total	338.622	256			
Corrected Total	12.734	255			

a. R Squared = .052 (Adjusted R Squared = .025)

Tests of Between-Subjects Effects

Dependent Variable: FinalFriction

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	2.576[a]	15	.172	4.057	.000
Intercept	296.661	1	296.661	7008.841	.000
Gel	.846	7	.121	2.855	.007
Gender	1.699	1	1.699	40.150	.000
Gel * Gender	.214	7	.031	.722	.654
Error	10.158	240	.042		
Total	338.622	256			
Corrected Total	12.734	255			

a. R Squared = .202 (Adjusted R Squared = .152)

Tests of Between-Subjects Effects

Dependent Variable: Normalized Final Friction

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	.547[a]	7	.078	2.845	.007
Intercept	254.704	1	254.704	9270.714	.000
Gel	.547	7	.078	2.845	.007
Error	6.731	245	.027		
Total	261.888	253			
Corrected Total	7.278	252			

a. R Squared = .075 (Adjusted R Squared = .049)

Tests of Between-Subjects Effects

Dependent Variable: Mean-Shifted Final Friction

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	.663[a]	7	.095	5.021	.000
Intercept	0.000	1	0.000	0.000	1.000
Gel	.663	7	.095	5.021	.000
Error	4.676	248	.019		
Total	5.339	256			
Corrected Total	5.339	255			

a. R Squared = .124 (Adjusted R Squared = .099)

Tests of Between-Subjects Effects

Dependent Variable: Mean-Shifted Normalized Final Friction

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	.544[a]	7	.078	4.016	.000
Intercept	2.260E-05	1	2.260E-05	.001	.973
Gel	.544	7	.078	4.016	.000
Error	4.737	245	.019		
Total	5.281	253			
Corrected Total	5.281	252			

a. R Squared = .103 (Adjusted R Squared = .077)

Tests of Between-Subjects Effects

Dependent Variable: UntreatedDynSkinStretch_m

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	.001[a]	7	.000	.885	.519
Intercept	.368	1	.368	2782.695	.000
Gel	.001	7	.000	.885	.519
Error	.032	245	.000		
Total	.402	253			
Corrected Total	.033	252			

a. R Squared = .025 (Adjusted R Squared = -.003)

Tests of Between-Subjects Effects

Dependent Variable: UntreatedDynSkinStretch_k

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	12213.964[a]	7	1744.852	.306	.951
Intercept	6125660.296	1	6125660.296	1072.993	.000
Gel	12213.964	7	1744.852	.306	.951
Error	1398691.661	245	5708.946		
Total	7539964.568	253			
Corrected Total	1410905.625	252			

a. R Squared = .009 (Adjusted R Squared = -.020)

Tests of Between-Subjects Effects

Dependent Variable: UntreatedDynSkinStretch_b

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	3.156[a]	7	.451	.630	.731
Intercept	280.218	1	280.218	391.361	.000
Gel	3.156	7	.451	.630	.731
Error	175.422	245	.716		
Total	459.071	253			
Corrected Total	178.577	252			

a. R Squared = .018 (Adjusted R Squared = -.010)

Tests of Between-Subjects Effects

Dependent Variable: RinsedDynSkinStretch_m

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	.003[a]	7	.000	1.263	.269
Intercept	.365	1	.365	1209.705	.000
Gel	.003	7	.000	1.263	.269
Error	.074	247	.000		
Total	.442	255			
Corrected Total	.077	254			

a. R Squared = .035 (Adjusted R Squared = .007)

Tests of Between-Subjects Effects

Dependent Variable: RinsedDynSkinStretch_k

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	26351.327[a]	7	3764.475	.597	.758
Intercept	6184763.866	1	6184763.866	980.580	.000
Gel	26351.327	7	3764.475	.597	.758
Error	1557890.772	247	6307.250		
Total	7771557.158	255			
Corrected Total	1584242.099	254			

a. R Squared = .017 (Adjusted R Squared = -.011)

Tests of Between-Subjects Effects

Dependent Variable: RinsedDynSkinStretch_b

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	.806[a]	7	.115	.872	.529
Intercept	235.710	1	235.710	1786.654	.000
Gel	.806	7	.115	.872	.529
Error	32.586	247	.132		
Total	269.150	255			
Corrected Total	33.392	254			

a. R Squared = .024 (Adjusted R Squared = -.004)

Tests of Between-Subjects Effects

Dependent Variable: FinalDynSkinStretch_m

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	.001[a]	7	.000	1.484	.173
Intercept	.390	1	.390	4292.930	.000
Gel	.001	7	.000	1.484	.173
Error	.023	248	9.096E-05		
Total	.414	256			
Corrected Total	.024	255			

a. R Squared = .040 (Adjusted R Squared = .013)

Tests of Between-Subjects Effects

Dependent Variable: FinalDynSkinStretch_k

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	32835.137[a]	7	4690.734	.633	.729
Intercept	7447973.414	1	7447973.414	1004.406	.000
Gel	32835.137	7	4690.734	.633	.729
Error	1838995.442	248	7415.304		
Total	9319803.993	256			
Corrected Total	1871830.579	255			

a. R Squared = .018 (Adjusted R Squared = -.010)

Tests of Between-Subjects Effects

Dependent Variable: FinalDynSkinStretch_b

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	2.695[a]	7	.385	.808	.582
Intercept	292.113	1	292.113	612.893	.000
Gel	2.695	7	.385	.808	.582
Error	118.200	248	.477		
Total	413.008	256			
Corrected Total	120.895	255			

a. R Squared = .022 (Adjusted R Squared = -.005)

Tests of Between-Subjects Effects

Dependent Variable: Viscous Damping Gel

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	101.782[a]	7	14.540	22.723	.000
Intercept	426.107	1	426.107	665.902	.000
Gel	101.782	7	14.540	22.723	.000
Error	134.378	210	.640		
Total	698.948	218			
Corrected Total	236.160	217			

a. R Squared = .431 (Adjusted R Squared = .412)

Tests of Between-Subjects Effects

Dependent Variable: Viscous Damping Foam

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	5.232[a]	7	.747	4.084	.000
Intercept	52.956	1	52.956	289.413	.000
Gel	5.232	7	.747	4.084	.000
Error	24.702	135	.183		
Total	89.743	143			
Corrected Total	29.933	142			

a. R Squared = .175 (Adjusted R Squared = .132)

APPENDIX B

C CODE FOR HYBRID POSITION/FORCE CONTROLLER AND MAGNETIC LEVITATION HAPTIC DEVICE MAIN PROGRAM

C Code for Hybrid Position/Force Controller from file ‘control.h’

```
void
HybridControl ( float * posActual, double * posDesired, double *
forceActual, double * forceDesired, double const hybridGains[][6],
double curr_time, double wrench[6], double forceActualCorrected[6],
           double rotAngles[3], double sVector[6], double dummy[][6])
{

      // Gain vector (rows: x, y, z, r_x, r_y, r_z)
      //             (cols: Kp, Kip, Kd, Kf, Kif, Kv)
      // r_x is the rotation about the x-axis
      // Kp, Kd, Kip are the PID gains for the position controller
      // Kf, Kv, Kif are the PIV gains for the force controller
      // FOR EXAMPLE:
      //    {{ KpX,   KiX,   KdX,   KfX,   KifX,   KvX };
      //     { KpY,   KiY,   KdY,   KfY,   KifY,   KvY };
      //     { KpZ,   KiZ,   KdZ,   KfZ,   KifZ,   KvZ };
      //     { KpR_X, KiR_X, KdR_X, KfR_X, KifR_X, KvR_X };
      //     { KpR_Y, KiR_Y, KdR_Y, KfR_Y, KifR_Y, KvR_Y };
      //     { KpR_Z, KiR_Z, KdR_Z, KfR_Z, KifR_Z, KvR_Z }};

      double kp[6];
      double ki[6];
      double kf[6];
      double kif[6];
      double kv[6];
      double posError[6];
      double forceError[6];
      double velocity[6];
      double velocityComp[6];
      double posIcomp[6];
      double posPcomp[6];
      double posPIcomp[6];
      double posPIcompS[6];
      double forceIcomp[6];
      double forcePcomp[6];
      double forcePIcomp[6];
      double forcePIcompIS[6];
      double PIDcomp[6];
      double gravityComp = 5.968;//constant

      static double curr_time_old = curr_time - .000000001;
      static double posIcompOld[6] = {0.0,0.0,0.0,0.0,0.0,0.0};

      static double forceIcompOld[6] = {0.0,0.0,0.0,0.0,0.0,0.0};

      double inverseSVector[6] = {1.0-sVector[0],1.0-sVector[1],1.0-
sVector[2],1.0-sVector[3],1.0-sVector[4],1.0-sVector[5]};

      double posActual3x1[3] = {0.0,0.0,0.0};// Don't need to make
angle 3x3 matrix, since we want them to be 0 no matter what frame
      double forceActual3x1[3] = {0.0,0.0,0.0};
```

```
	double zero2ConstraintRot[3][3] =
{{cos(rotAngles[1]),0.0,sin(rotAngles[1])},{0.0,1.0,0.0},{-
sin(rotAngles[1]),0.0,cos(rotAngles[1])}};
	double constraint2ZeroRot[3][3] = {{cos(-rotAngles[1]),0.0,sin(-
rotAngles[1])},{0.0,1.0,0.0},{-sin(-rotAngles[1]),0.0,cos(-
rotAngles[1])}};
	double posActual3x1rot[3] = {0.0,0.0,0.0};
	double forceActual3x1rot[3] = {0.0,0.0,0.0};
	double posDesired3x1[3] = {0.0,0.0,0.0};
	double forceDesired3x1[3] = {0.0,0.0,0.0};
	double PIDcomp3x1[3] = {0.0,0.0,0.0};
	double PIDcomp3x1zero[3] = {0.0,0.0,0.0};

	double delta_t = curr_time - curr_time_old;
	curr_time_old = curr_time;

	if (delta_t == 0.0)
	{
		delta_t = 0.0000000000001;//0.00000000001;
	}

	for (int i = 0; i < 6; i++)
	{
		for (int j = 0; j < 6; j++)
		{
			dummy[i][j] = 0.0;
		}

		kp[i] = 0.0;
		ki[i] = 0.0;
		kf[i] = 0.0;
		kif[i] = 0.0;
		kv[i] = 0.0;
		posError[i] = 0.0;
		forceError[i] = 0.0;
		velocity[i] = 0.0;
		velocityComp[i] = 0.0;
		posIcomp[i] = 0.0;
		posPcomp[i] = 0.0;
		posPIcomp[i] = 0.0;
		posPIcompS[i] = 0.0;
		forceIcomp[i] = 0.0;
		forcePcomp[i] = 0.0;
		forcePIcomp[i] = 0.0;
		forcePIcompIS[i] = 0.0;
		PIDcomp[i] = 0.0;

		kp[i] = hybridGains[i][0];
		ki[i] = hybridGains[i][1];
		//kd column 2 not used
		kf[i] = hybridGains[i][3];
		kif[i] = hybridGains[i][4];
		kv[i] = hybridGains[i][5];
		wrench[i] = 0.0;
		forceActualCorrected[i] = 0.0;
	}
```

```
      for (int i = 0; i < 3; i++) // make diagonal 3x3 matrices from
3x1 vectors
      {
            posActual3x1[i] = posActual[i];
            forceActual3x1[i] = forceActual[i];
            posDesired3x1[i] = posDesired[i];
            forceDesired3x1[i] = forceDesired[i];
      }

      MatrixMultiply3x3x3x1(zero2ConstraintRot,posActual3x1,posActual3x
1rot);// 3x3
      MatrixMultiply3x3x3x1(zero2ConstraintRot,forceActual3x1,forceActu
al3x1rot);

      for ( int p = 0; p < 3; p++ ) // Break down rotated force values
into  x, y and z components for output forceActualCorrected. Note that
torque values are not rotated
      {
            forceActualCorrected[p] = forceActual3x1rot[p];
            forceActualCorrected[p+3] = forceActual[p+3];
      }

      static double position6x1Old[6] =
{posActual3x1rot[0],posActual3x1rot[1],posActual3x1rot[2],posActual[3],
posActual[4],posActual[5]};

      double position6x1[6] =
{posActual3x1rot[0],posActual3x1rot[1],posActual3x1rot[2],posActual[3],
posActual[4],posActual[5]};

      for (int i = 0; i < 6; i++)
      {
            velocity[i] = (position6x1[i]-position6x1Old[i])/delta_t;
            position6x1Old[i] = position6x1[i];
      }

      for (int i = 0; i < 3; i++)
      {
            posError[i] = posDesired3x1[i] - posActual3x1rot[i];
            forceError[i] = forceDesired3x1[i] - forceActual3x1rot[i];
      }

      for (int i = 3; i < 6; i++)
      {
            posError[i] = posDesired[i] - posActual[i]; // this is for
the x, y, and z angles or orientation.
            //Since we are controlling them through position, and we
know we want them to stay at 0, then we don't
            //need to rotate them, and thus we only need the last three
negative (desired = 0) of the actual position.
      }

      VectorMultiply6(kp,posError,posPcomp); // 6x6
      VectorMultiply6(ki,posError,posIcomp);
      VectorMultiply6(kf,forceError,forcePcomp);
      VectorMultiply6(kif,forceError,forceIcomp);
      VectorMultiply6(kv,velocity,velocityComp);
```

```
	for (int i = 0; i < 6; i++)
	{
		posIcomp[i] += posIcompOld[i];
		forceIcomp[i] += forceIcompOld[i];
		posIcompOld[i] = posIcomp[i];
		forceIcompOld[i] = forceIcomp[i];
		posPIcomp[i] = posPcomp[i] + posIcomp[i]*delta_t;
		forcePIcomp[i] = forcePcomp[i] + forceIcomp[i]*delta_t;
	}

	VectorMultiply6(sVector,posPIcomp,posPIcompS);
	VectorMultiply6(inverseSVector,forcePIcomp,forcePIcompIS);

	for (int i = 0; i < 6; i++)
	{
		PIDcomp[i] = posPIcompS[i] + forcePIcompIS[i] -
velocityComp[i];
	}

	PIDcomp3x1[0] = PIDcomp[0] + (gravityComp*(sin(rotAngles[1])));
// gravity compensation
	PIDcomp3x1[1] = PIDcomp[1];
	PIDcomp3x1[2] = PIDcomp[2] + (gravityComp*(cos(rotAngles[1])));
// gravity compensation

	MatrixMultiply3x3x3x1(constraint2ZeroRot,PIDcomp3x1,PIDcomp3x1zer
o);

	for (int i = 0; i < 3; i++)
	{
		PIDcomp[i] = PIDcomp3x1zero[i];
	}

	for (int i = 0; i < 6; i++)
	{
		wrench[i]=PIDcomp[i];
	}

} // HybridControl
//////////////////////////////////////////////////////////////////////

Matrix Multiply Functions
//////////////////////////////////////////////////////////////////////
// Multiply two matrices matA*matB=matC of known dimensions, given
inputs for numbers of rows & columns each and the matrices themselves
void
MatrixMultiply6x6 ( double matA[6][6], double matB[6][6], double
matC[6][6] )
{
	int rowsA = 6;
	int colsA = 6;
	int rowsB = 6;
	int colsB = 6;

	for( int i = 0; i < rowsA; i++ )
	{
```

```
            for( int j=0; j < colsB; j++ )
            {
                  matC[i][j] = 0;
                  for( int k = 0; k < colsA; k++)
                  {
                        matC[i][j] += matA[i][k]*matB[k][j];
                  }
            }
      }
}

// Multiply two matrices matA*matB=matC of known dimensions, given
inputs for numbers of rows & columns each and the matrices themselves
void
MatrixMultiply3x3 ( double matA[3][3], double matB[3][3], double
matC[3][3] )
{
      int rowsA = 3;
      int colsA = 3;
      int rowsB = 3;
      int colsB = 3;

      for( int i = 0; i < rowsA; i++ )
      {
            for( int j=0; j < colsB; j++ )
            {
                  matC[i][j] = 0;
                  for( int k = 0; k < colsA; k++)
                  {
                        matC[i][j] += matA[i][k]*matB[k][j];
                  }
            }
      }
}

// Multiply two 6x1 vectors element-wise vectA*vectB=vectC of known
dimensions, given the vectors themselves
void
VectorMultiply6 ( double vectA[6], double vectB[6], double vectC[6] )
{
      for( int i = 0; i < 6; i++ )
      {
            vectC[i] = 0;
            vectC[i] = vectA[i]*vectB[i];
      }
}

// Multiply two matrices matA*matB=matC of known dimensions, given
inputs for numbers of rows & columns each and the matrices themselves
void
MatrixMultiply3x3x3x1 ( double matA[3][3], double matB[3], double
matC[3] )
{
      int rowsA = 3;
      int colsA = 3;
      int rowsB = 3;
      int colsB = 1;
```

```
	for( int i = 0; i < rowsA; i++ )
	{
		for( int j=0; j < colsB; j++ )
		{
			matC[i] = 0;
			for( int k = 0; k < colsA; k++)
			{
				matC[i] += matA[i][k]*matB[k];
			}
		}
	}
}
```

Main program C code from file ‘main.c’

```
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <comedilib.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <getopt.h>
#include <ctype.h>
#include <time.h>
#include <sys/time.h>
#include <math.h>

#define dataPoints 120000
#define numDataColumns 46//38//32

#include "control.h"

// Time step to use for regulating the loop
const int time_step = 1; // milliseconds
time_t start,end;

double record_Data[dataPoints][numDataColumns];
double record_Data_skinStretch[dataPoints][numDataColumns];
double record_DataWet[dataPoints][numDataColumns];


int
main(int argc, char *argv[])
{
	unsigned long long int time_start, time_finish, time_current,
time_next, ticks_per_millisecond, tStart;
	double time_current_seconds;

	// Determine internal clock speed
	printf("Determining internal clock speed...\n");
	ticks_per_millisecond = find_ticks_per_millisecond();
```

```
        printf("Start Time = %lld\tTicks per millisecond = %lld\n",
time_start, ticks_per_millisecond);
      time_next = time_step*ticks_per_millisecond;


      double comedi_voltage[DATASIZE];
      double maglev_position[DATASIZE];
      long long int curr_time;
      long long int click_sec;

      double curr_time_double;
      double time_next_double;

      double force_curr[DATASIZE];
      double force_des[DATASIZE];

      bool flags[DATASIZE] ;


//    double zVelocity = 1.0;// units mm/sec
//    double zStep = (zVelocity*1.0/1000.0)/1000.0;
//*time_slice*8)/1000.0;// use zVelocity and time_slice to calculate
zStep and convert it into units of METERS
      double xVelocity = 2.0;// units mm/sec
      double xStep =
(xVelocity*1.0/1000.0)/1000.0;//*2.0//*time_slice*8)/1000.0; // use
xVelocity and time_slice to calculate xStep and convert it into units
of METERS
      static double nextStep[] = {0.0,0.0,0.0,0.0,0.0,0.0};
      static double forceActualFiltered[6] = {0.0,0.0,0.0,0.0,0.0,0.0};
      static double forceActualCorrected[6] =
{0.0,0.0,0.0,0.0,0.0,0.0};
      static double force_zero[6];

      char year[6];
      char month[6];
      char date[6];
      char hours[6];
      char minutes[6];
      char seconds[6];
//printf("time_finish = %llu \n",time_finish);

////////////////////////////////////////////////////////////////////////
////
//    Main test Parameters -- To be edited before beginning testing
      ///
//                                                        ///
//          Other test parameters can be changed when prompted    ///
//          to do so in the command prompt                        ///
////////////////////////////////////////////////////////////////////////
////
////////////////////////////////////////////////////////////////////////
////
////////////////////////////////////////////////////////////////////////
///
                                                          ///
      int subjectNum = 32;
```

```
      char test =       'B';
                  //'B';
////////////////////////////////////////////////////////////////////////
////
////////////////////////////////////////////////////////////////////////
////
////////////////////////////////////////////////////////////////////////
////

      bool dryStrokingTest = true;
      bool skinStretchTest = true;// Default    unless changed later
      bool overwriteFilenames = false;
      int wetPrepTimeSeconds = 60;
      int foamPrepTimeSeconds = 30;
      bool detailed_instructions_and_timing = true;

      getDateTimeStr ( year, month, date, hours, minutes, seconds );
      initializeRecordDateTimeStr ( year, month, date, hours, minutes,
seconds, subjectNum, test );
      int gelOrderRow = (subjectNum-1) % 8;

      int gelOrder[8][8] = {
      {1,2,8,3,7,4,6,5},//{2,6,7,4,5,1,8,3},//{1,1,7,7,7,4,6,5},

      {2,3,1,4,8,5,7,6},//{4,7,6,2,8,5,7,6},//{7,7,1,1,8,5,7,6},
                        {3,4,2,5,1,6,8,7},//{7,4,2,6,1,3,5,8},
                        {4,5,3,6,2,7,1,8},//{1,2,5,6,2,7,1,8},
                        {5,6,4,7,3,8,2,1},
                        {6,7,5,8,4,1,3,2},
                        {7,8,6,1,5,2,4,3},
                        {8,1,7,2,6,3,5,4}};
      int a;
      int currGelOrder[4];

      if (test == 'A')
      {
            for ( a = 0; a<4; a++ )
            {
                  currGelOrder[a] = gelOrder[gelOrderRow][a];// Changed
from subjectNum-1 to gelOrderRow on Jul 3
                  printf("\ncurrGelOrder = %i\n",currGelOrder[a]);
            }
      }
      else if (test == 'B')
      {
            for ( a = 0; a<4; a++ )
            {
                  currGelOrder[a] = gelOrder[gelOrderRow][a+4];
                  printf("\ncurrGelOrder = %i\n",currGelOrder[a]);
            }
      }


      bool noPositionRot = true; // if true, no position-controlled
rotation about the y-axis
      double thetaY = -0.058643;//initRotGoal = 0.058643;   // radians
                        ///
```

```
      double rotAngles[3] = {0.0,thetaY,0.0};//{-PI/4.0,0.0,0.0};// in
radians, about x, y, and z axis, respectively

////////////////////////////////////////////////////////////////////////
////

      char showerGelList[8][20] =
{"Dial","DoveDeepMoisture","DoveGoFresh","Olay","SSNutriSerums","SSPure
Cashmere","SSSheaButter","SweetPeaGel"};//ShowerGel Type
      char testLocationList[4][10] =
{"LeftDist","LeftProx","RightDist","RightProx"};//Test Location
      char moistureStateList[5][6] =
{"Dry","Gel","Foam","Rinse","ReDry"};// Skin Moisture State
//    char dryProductStateList[3][6] = {"Dry","Rinse","ReDry"};// Skin
Moisture State
//    char wetProductStateList[2][6] = {"Gel","Foam"};
      char *gel = new char[20];
      char *loc = new char[10];
      char *state = new char[6];

      gel = showerGelList[0];
      loc = testLocationList[0];
      state = moistureStateList[0];

      double desNormForce = -0.1;                              ///
      double positiveLimit = 8.0/1000.0; // units mm; distance from
origin along positive x axis at which stroking stops
      double negativeLimit = -8.0/1000.0; // units mm; distance from
origin along negative x axis at which stroking starts
      bool WITH_MAGLEV = true;                                 ///
      bool WITH_TRAJ = true;                                   ///
                                  ///
                                                        ///
////////////////////////////////////////////////////////////////////////
////


      ml_position_t cur_pos;
      ml_forces_t cur_force;
      ml_forces_t des_force;


 // Initialize the comedi
      comedi_t *device;
      ml_device_handle_t *device_handler;
      device_handler = new ml_device_handle_t;
      ml_gainset_type_t mpGainSetType;
      ml_gain_vec_t gainVec;

      static SHARED_VAR *conf;
      int *shm_fd;

      conf = new SHARED_VAR;
      shm_fd = new int;

      //Open the device comedi
      device = comedi_open ( device_name );
```

```
      if ( !device )
      {
            comedi_perror ( device_name );
            exit ( -1 );
      }

       //Connect the maglev
      if ( WITH_MAGLEV )
      {

//          magLevInitialize ( device_handler, des_force, cur_force,
cur_pos, mpGainSetType, gainVec, xStep);

            MaglevConnect ( maglev_server_name, device_handler );

      //Set the gain
            mpGainSetType = ML_GAINSET_TYPE_NORMAL;

            for (int i=0; i<6; i++)
            {
                  gainVec.values[i].p  = default_gains[i][0];
                  gainVec.values[i].i  = default_gains[i][1];
                  gainVec.values[i].d  = default_gains[i][2];
                  gainVec.values[i].ff = 0.0;
                  printf("Setting the gain, %f, %f, %f\n",
                  gainVec.values[i].p, gainVec.values[i].i,
gainVec.values[i].d );
            }
            gainVec.values[2].ff = 3.5;
            ml_SetGainVecAxes ( *device_handler, mpGainSetType, gainVec
);


      // Check actual position for moving the flotor to the desired x
position
            ml_GetActualPosition( *device_handler, &cur_pos );


      // Display current wrench and request user action and input
            ml_GetForces( *device_handler, &cur_force );

            printf("xStep = %e \n", xStep);
            printf("Current Wrench =
%e,%e,%e,%e,%e,%e\n",cur_force.values[0],cur_force.values[1],cur_force.
values[2],cur_force.values[3],cur_force.values[4],cur_force.values[5]);

            printf("\nPre force_zero: %f\n",force_zero[2]);
            retractAndZeroForceSensor ( device_handler, device,
cur_pos, nextStep, force_zero, comedi_voltage );
            printf("\nPost force_zero: %f\n",force_zero[2]);

      }

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
////////////
```

```
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
////////////

//printf("\ngel= %s",showerGelList[gelOrder[gelOrderRow][0]-1]);
//printf("\ngel= %s",showerGelList[currGelOrder[0]-1]);
//printf("\ncurrGelOrder= %i",currGelOrder[0]);

int menuNum;
//bool breakFlag = false;

testMenu:
//breakFlag = false;
printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\nTest Subject:
%i%c\n\nShower gels used in this
test:\t%s,\t%s,\t%s,\t%s\n\n",subjectNum,test,showerGelList[currGelOrde
r[0]-1],showerGelList[currGelOrder[1]-1],showerGelList[currGelOrder[2]-
1],showerGelList[currGelOrder[3]-1]);//\n\n\n\n\n\n
printf("\n\nTEST MENU: \nPlease enter a number 1 through 24
corresponding to when in the test you wish to begin... OR [0] to
exit\n\n");
printf("\t 1: Dry\t\tLeft Distal\n\t 2: Dry\t\tLeft Proximal\n\t 3:
Dry\t\tRight Distal\n\t 4: Dry\t\tRight Proximal\n");
printf("\t 5: Gel\t\tLeft Distal\n\t 6: Foam\tLeft Distal\n\t 7:
Rinse\tLeft Distal\n\t 8: Gel\t\tLeft Proximal\n");
printf("\t 9: Foam\tLeft Proximal\n\t 10: Rinse\tLeft Proximal\n\t 11:
Wait\tLeft Distal\n\t 12: ReDry\tLeft Distal\n");
printf("\t 13: Gel\tRight Distal\n\t 14: Foam\tRight Distal\n\t 15:
Rinse\tRight Distal\n\t 16: Wait\tLeft Proximal\n\t 17: ReDry\tLeft
Proximal\n");
printf("\t 18: Gel\tRight Proximal\n\t 19: Foam\tRight Proximal\n\t 20:
Rinse\tRight Proximal\n\t 21: Wait\tRight Distal\n\t 22: ReDry\tRight
Distal\n");
printf("\t 23: Wait\tRight Proximal\n\t 24: ReDry\tRight
Proximal\n\n");

std::cin >> menuNum;

//printf("%i",menuNum);


////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
//////////////
int numSeconds = 10;
int userResponse;
int continueResponse;
int currLoc;
int stateNum;
int stepNum;

double goalTime = (20.0*60.0);//(20.0*60.0);// 20 minutes converted to
seconds

double diff_loc1;
double waitTime_loc1;
double waitTimeOld_loc1;
```

```
int waitTimeInt_loc1;
int waitTimeIntOld_loc1;

int waitTimeSeconds_loc1;

double diff_loc2;
double waitTime_loc2;
double waitTimeOld_loc2;
int waitTimeInt_loc2;
int waitTimeIntOld_loc2;

double diff_loc3;
double waitTime_loc3;
double waitTimeOld_loc3;
int waitTimeInt_loc3;
int waitTimeIntOld_loc3;

double diff_loc4;
double waitTime_loc4;
double waitTimeOld_loc4;
int waitTimeInt_loc4;
int waitTimeIntOld_loc4;

int secondsAtRinse = 0;
int secondsSinceRinse = 0;
int secondsCurrent = 0;
int minutesSinceRinse = 0;

int wetPrepTimer;
int wetPrepTimerOld;
int foamPrepTimer;
int foamPrepTimerOld;

int userResponse5a;
int userResponse5b;
int userResponse6a;
int userResponse6b;
int userResponse7a;

int userResponse8a;
int userResponse8b;
int userResponse9a;
int userResponse9b;
int userResponse10a;

int userResponse13a;
int userResponse13b;
int userResponse14a;
int userResponse14b;
int userResponse15a;

int userResponse18a;
int userResponse18b;
int userResponse19a;
int userResponse19b;
int userResponse20a;
```

```
//int caseNum;
//bool breakFlag = false;

switch (menuNum) {

      case 1:// Left Distal Dry 1,1

            stepNum = 1;

            currLoc = 1;//first case = first loc
            stateNum = 1;//first state = dry

            printf("\nStep %i: %s - Location
%i\n",stepNum,state,currLoc);

            gel = showerGelList[currGelOrder[currLoc-1]-1];// first gel
            loc = testLocationList[currLoc-1];// first test location
            state = moistureStateList[stateNum-1];// first state =
initial dry state

            usleep(2000000);
            printf("\nPre force_zero: %f\n",force_zero[2]);
            retractAndZeroForceSensor ( device_handler, device,
cur_pos, nextStep, force_zero, comedi_voltage );
            printf("\nPost force_zero: %f\n",force_zero[2]);

            reDoForceDisplay1:
            forceDisplay ( device, comedi_voltage, force_curr,
ticks_per_millisecond, forceActualFiltered, force_zero, numSeconds );

            // Ask for user response, act accordingly
            loc1_dry:
            printf("\nStep 1 of 24... %s forearm, %s gel,
%s\n",loc,gel,state);
            printf("Do you wish to exit [0] OR continue [1] OR repeat
normal force printout [2] OR return to test menu [3] ?\n");
            std::cin >> userResponse;
            while ( userResponse != 0 && userResponse!=1 &&
userResponse!=2 && userResponse!=3) std::cin >> userResponse;
            if ( userResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( userResponse==2 ) goto reDoForceDisplay1;
            else if ( userResponse==3 )
            {
//                breakFlag = true;
//                break;
                  goto testMenu;
            }
            else if (userResponse != 0 && userResponse != 1 &&
userResponse != 2 && userResponse != 3 )
            {
                  printf(" \n--Incorrect Entry--\n");
                  goto loc1_dry;
            }
            else if ( userResponse==1 )
            {
```

```
                  DryStrokingSkinStretch ( device_handler, device,
cur_pos, cur_force, des_force, ticks_per_millisecond, time_next,
time_step, time_current_seconds, record_Data, record_Data_skinStretch,
xStep, dryStrokingTest, skinStretchTest, noPositionRot, rotAngles,
force_zero, positiveLimit, negativeLimit, WITH_MAGLEV, WITH_TRAJ,
comedi_voltage, force_curr, desNormForce, thetaY, nextStep,
forceActualFiltered, forceActualCorrected, subjectNum, state, gel, loc,
test, stepNum, overwriteFilenames );
            }

            printf("\nDo you wish to [0] exit OR [1] continue to Step 2
( Location 2 - Dry ) OR [2] return to Test Menu ?\n");
            std::cin >> continueResponse;
            if ( continueResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( continueResponse == 2 ) goto testMenu;


      case 2:// Left Proximal Dry 2,1

            stepNum = 2;

            currLoc = 2;
            stateNum = 1;
            gel = showerGelList[currGelOrder[currLoc-1]-1];// first gel
            loc = testLocationList[currLoc-1];// first test location
            state = moistureStateList[stateNum-1];// first state =
initial dry state

            printf("\nStep %i: %s - Location
%i\n",stepNum,state,currLoc);

            usleep(2000000);
            printf("\nPre force_zero: %f\n",force_zero[2]);
            retractAndZeroForceSensor ( device_handler, device,
cur_pos, nextStep, force_zero, comedi_voltage );
            printf("\nPost force_zero: %f\n",force_zero[2]);

            reDoForceDisplay2:
            forceDisplay ( device, comedi_voltage, force_curr,
ticks_per_millisecond, forceActualFiltered, force_zero, numSeconds );

            // Ask for user response, act accordingly
            loc2_dry:
            printf("\nStep 2 of 24... %s forearm, %s gel,
%s\n",loc,gel,state);
            printf("Do you wish to exit [0] OR continue [1] OR repeat
normal force printout [2] OR return to test menu [3] ?\n");
            std::cin >> userResponse;
            while ( userResponse != 0 && userResponse!=1 &&
userResponse!=2 && userResponse!=3) std::cin >> userResponse;
            if ( userResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( userResponse==2 ) goto reDoForceDisplay2;
            else if ( userResponse==3 )
            {
                  goto testMenu;
```

```
            }
            else if (userResponse != 0 && userResponse != 1 &&
userResponse != 2 && userResponse != 3 )
            {
                  printf(" \n--Incorrect Entry--\n");
                  goto loc2_dry;
            }
            else if ( userResponse==1 )
            {
                  DryStrokingSkinStretch ( device_handler, device,
cur_pos, cur_force, des_force, ticks_per_millisecond, time_next,
time_step, time_current_seconds, record_Data, record_Data_skinStretch,
xStep, dryStrokingTest, skinStretchTest, noPositionRot, rotAngles,
force_zero, positiveLimit, negativeLimit, WITH_MAGLEV, WITH_TRAJ,
comedi_voltage, force_curr, desNormForce, thetaY, nextStep,
forceActualFiltered, forceActualCorrected, subjectNum, state, gel, loc,
test, stepNum, overwriteFilenames );
            }

            printf("\nDo you wish to [0] exit OR [1] continue to Step 3
( Location 3 - Dry ) OR [2] return to Test Menu ?\n");
            std::cin >> continueResponse;
            if ( continueResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( continueResponse == 2 ) goto testMenu;

      case 3:// Right Distal Dry 3,1

            stepNum = 3;

            currLoc = 3;
            stateNum = 1;
            gel = showerGelList[currGelOrder[currLoc-1]-1];// first gel
            loc = testLocationList[currLoc-1];// first test location
            state = moistureStateList[stateNum-1];// first state =
initial dry state

            printf("\nStep %i: %s - Location
%i\n",stepNum,state,currLoc);

            usleep(2000000);
            printf("\nPre force_zero: %f\n",force_zero[2]);
            retractAndZeroForceSensor ( device_handler, device,
cur_pos, nextStep, force_zero, comedi_voltage );
            printf("\nPost force_zero: %f\n",force_zero[2]);

            reDoForceDisplay3:
            forceDisplay ( device, comedi_voltage, force_curr,
ticks_per_millisecond, forceActualFiltered, force_zero, numSeconds );

            // Ask for user response, act accordingly
            loc3_dry:
            printf("\nStep 3 of 24... %s forearm, %s gel,
%s\n",loc,gel,state);
            printf("Do you wish to exit [0] OR continue [1] OR repeat
normal force printout [2] OR return to test menu [3] ?\n");
            std::cin >> userResponse;
```

```
            while ( userResponse != 0 && userResponse!=1 &&
userResponse!=2 && userResponse!=3) std::cin >> userResponse;
            if ( userResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( userResponse==2 ) goto reDoForceDisplay3;
            else if ( userResponse==3 )
            {
                  goto testMenu;
            }
            else if (userResponse != 0 && userResponse != 1 &&
userResponse != 2 && userResponse != 3 )
            {
                  printf(" \n--Incorrect Entry--\n");
                  goto loc3_dry;
            }
            else if ( userResponse==1 )
            {
                  DryStrokingSkinStretch ( device_handler, device,
cur_pos, cur_force, des_force, ticks_per_millisecond, time_next,
time_step, time_current_seconds, record_Data, record_Data_skinStretch,
xStep, dryStrokingTest, skinStretchTest, noPositionRot, rotAngles,
force_zero, positiveLimit, negativeLimit, WITH_MAGLEV, WITH_TRAJ,
comedi_voltage, force_curr, desNormForce, thetaY, nextStep,
forceActualFiltered, forceActualCorrected, subjectNum, state, gel, loc,
test, stepNum, overwriteFilenames );
            }

            printf("\nDo you wish to [0] exit OR [1] continue to Step 4
( Location 4 - Dry ) OR [2] return to Test Menu ?\n");
            std::cin >> continueResponse;
            if ( continueResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( continueResponse == 2 ) goto testMenu;

      case 4:// Right Proximal Dry 4,1

            stepNum = 4;

            currLoc = 4;
            stateNum = 1;
            gel = showerGelList[currGelOrder[currLoc-1]-1];// first gel
            loc = testLocationList[currLoc-1];// first test location
            state = moistureStateList[stateNum-1];// first state =
initial dry state

            printf("\nStep %i: %s - Location
%i\n",stepNum,state,currLoc);

            usleep(2000000);
            printf("\nPre force_zero: %f\n",force_zero[2]);
            retractAndZeroForceSensor ( device_handler, device,
cur_pos, nextStep, force_zero, comedi_voltage );
            printf("\nPost force_zero: %f\n",force_zero[2]);

            reDoForceDisplay4:
            forceDisplay ( device, comedi_voltage, force_curr,
ticks_per_millisecond, forceActualFiltered, force_zero, numSeconds );
```

```
            // Ask for user response, act accordingly
            loc4_dry:
            printf("\nStep 4 of 24... %s forearm, %s gel,
%s\n",loc,gel,state);
            printf("Do you wish to exit [0] OR continue [1] OR repeat
normal force printout [2] OR return to test menu [3] ?\n");
            std::cin >> userResponse;
            while ( userResponse != 0 && userResponse!=1 &&
userResponse!=2 && userResponse!=3) std::cin >> userResponse;
            if ( userResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( userResponse==2 ) goto reDoForceDisplay4;
            else if ( userResponse==3 )
            {
                  goto testMenu;
            }
            else if (userResponse != 0 && userResponse != 1 &&
userResponse != 2 && userResponse != 3 )
            {
                  printf(" \n--Incorrect Entry--\n");
                  goto loc4_dry;
            }
            else if ( userResponse==1 )
            {
                  DryStrokingSkinStretch ( device_handler, device,
cur_pos, cur_force, des_force, ticks_per_millisecond, time_next,
time_step, time_current_seconds, record_Data, record_Data_skinStretch,
xStep, dryStrokingTest, skinStretchTest, noPositionRot, rotAngles,
force_zero, positiveLimit, negativeLimit, WITH_MAGLEV, WITH_TRAJ,
comedi_voltage, force_curr, desNormForce, thetaY, nextStep,
forceActualFiltered, forceActualCorrected, subjectNum, state, gel, loc,
test, stepNum, overwriteFilenames );
            }

            printf("\nDo you wish to [0] exit OR [1] continue to Step 5
( Location 1 - Gel ) OR [2] return to Test Menu ?\n");
            std::cin >> continueResponse;
            if ( continueResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( continueResponse == 2 ) goto testMenu;

      case 5:// Left Distal Gel 1,2

            stepNum = 5;

            currLoc = 1;
            stateNum = 2;
            gel = showerGelList[currGelOrder[currLoc-1]-1];// first gel
            loc = testLocationList[currLoc-1];// first test location
            state = moistureStateList[stateNum-1];// first state =
initial dry state

            printf("\nStep %i: %s %s - Location
%i\n",stepNum,gel,state,currLoc);

            if ( detailed_instructions_and_timing )
```

```
            {
                  printf("(a) Prepare arm-rest spacers so that tactor
touches lightly on Location %d\n",currLoc);
                  printf("(b) Put a new finger cot onto your right
index finger\n");
                  printf("(c) Retrieve the soaked sponge, and without
wringing it out, place it over the oval at Location %d\n",currLoc);
                  printf("(d) Let it rest on Location %d for a period
of %i seconds...\n\nSelect [1] to begin
timer\n",currLoc,wetPrepTimeSeconds);
                  std::cin >> userResponse5a;
                  while ( userResponse5a != 1 ) std::cin >>
userResponse5a;
                  if ( userResponse5a == 1 )
                  {
                        time(&start);
                        time(&end);
                        wetPrepTimer = difftime(end,start);
                        //printf("wetPrepTimer = %d",wetPrepTimer);
                        while ( wetPrepTimer < wetPrepTimeSeconds )
                        {
                              time(&end);
                              wetPrepTimer = difftime(end,start);
                              if ( wetPrepTimer != wetPrepTimerOld )
                              {
                                    printf("%i seconds
remaining...\n",(wetPrepTimeSeconds-wetPrepTimer));
                                    wetPrepTimerOld = wetPrepTimer;
                              }
                        }
                        printf("\n(e) Remove cloth from arm, and place
it back into the clean container\n");
                  }
                  //else if ( userResponse == 2 ) goto TestMenu;

                  printf("(f) Apply 1 mL of '%s' shower gel to the oval
at Location %d\n",gel,currLoc);
                  printf("(g) Ensure shower gel covers the entire oval
spread it around gently if needed\n");
                  printf("(h) Place arm face down onto the tactor with
the oval centered on the tactor, and adjust the spacers if needed\n");
                  printf("(i) Keep the finger cot on your finger for
now\n\nSelect [1] to continue...\n");

                  std::cin >> userResponse5b;
                  while ( userResponse5b != 1 ) std::cin >>
userResponse5b;
            }

            usleep(2000000);
            printf("\nPre force_zero: %f\n",force_zero[2]);
            retractAndZeroForceSensor ( device_handler, device,
cur_pos, nextStep, force_zero, comedi_voltage );
            printf("\nPost force_zero: %f\n",force_zero[2]);

            reDoForceDisplay5:
```

```
            forceDisplay ( device, comedi_voltage, force_curr,
ticks_per_millisecond, forceActualFiltered, force_zero, numSeconds );

            // Ask for user response, act accordingly
            loc5_dry:
            printf("\nStep 5 of 24... %s forearm, %s gel,
%s\n",loc,gel,state);
            printf("Do you wish to exit [0] OR continue [1] OR repeat
normal force printout [2] OR return to test menu [3] ?\n");
            std::cin >> userResponse;
            while ( userResponse != 0 && userResponse!=1 &&
userResponse!=2 && userResponse!=3) std::cin >> userResponse;
            if ( userResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( userResponse==2 ) goto reDoForceDisplay5;
            else if ( userResponse==3 )
            {
                  goto testMenu;
            }
            else if (userResponse != 0 && userResponse != 1 &&
userResponse != 2 && userResponse != 3 )
            {
                  printf(" \n--Incorrect Entry--\n");
                  goto loc5_dry;
            }
            else if ( userResponse==1 )
            {
                  WetStroking ( device_handler, device,
ticks_per_millisecond, time_next, time_step, time_current_seconds,
xStep, noPositionRot, rotAngles, force_zero, positiveLimit,
negativeLimit, WITH_MAGLEV, WITH_TRAJ, comedi_voltage, force_curr,
desNormForce, thetaY, nextStep, forceActualFiltered,
forceActualCorrected, subjectNum, state, gel, loc, record_DataWet,
test, stepNum, overwriteFilenames );
            }

            printf("\nDo you wish to [0] exit OR [1] continue to Step 6
( Location 1 - Foam ) OR [2] return to Test Menu ?\n");
            std::cin >> continueResponse;
            if ( continueResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( continueResponse == 2 ) goto testMenu;

      case 6:// Left Distal Foam 1,3

            stepNum = 6;

            currLoc = 1;
            stateNum = 3;
            gel = showerGelList[currGelOrder[currLoc-1]-1];// first gel
            loc = testLocationList[currLoc-1];// first test location
            state = moistureStateList[stateNum-1];// first state =
initial dry state

            printf("\nStep %i: %s %s - Location
%i\n",stepNum,gel,state,currLoc);
```

```
            if ( detailed_instructions_and_timing )
            {
                  printf("(a) Get ready to lather the shower gel at
Location %d by rubbing vigorously for %d
seconds\n",currLoc,foamPrepTimeSeconds);
                  printf("(b) Select [1] to start the %d second
timer.\nTest Subject: begin lathering\nTest Administrator: clean
tactor\n",foamPrepTimeSeconds);
                  std::cin >> userResponse6a;
                  while ( userResponse6a != 1 ) std::cin >>
userResponse6a;
                  if ( userResponse6a == 1 )
                  {
                        time(&start);
                        time(&end);
                        foamPrepTimer = difftime(end,start);
                        //printf("wetPrepTimer = %d",wetPrepTimer);
                        while ( foamPrepTimer < foamPrepTimeSeconds )
                        {
                              time(&end);
                              foamPrepTimer = difftime(end,start);
                              if ( foamPrepTimer != foamPrepTimerOld )
                              {
                                    printf("%i seconds
remaining...\n",(foamPrepTimeSeconds-foamPrepTimer));
                                    foamPrepTimerOld = foamPrepTimer;
                              }
                        }
                  }
                  //else if ( userResponse == 2 ) goto TestMenu;

                  printf("(c) Remove finger cot\n");
                  printf("(d) Again place arm face down onto the tactor
with the Location %d oval centered on the tactor, and adjust the
spacers so that the arm is touching lightly\n",currLoc);
                  printf("\nSelect [1] to continue...\n");

                  std::cin >> userResponse6b;
                  while ( userResponse6b != 1 ) std::cin >>
userResponse6b;
            }

            usleep(2000000);
            printf("\nPre force_zero: %f\n",force_zero[2]);
            retractAndZeroForceSensor ( device_handler, device,
cur_pos, nextStep, force_zero, comedi_voltage );
            printf("\nPost force_zero: %f\n",force_zero[2]);

            reDoForceDisplay6:
            forceDisplay ( device, comedi_voltage, force_curr,
ticks_per_millisecond, forceActualFiltered, force_zero, numSeconds );

            // Ask for user response, act accordingly
            loc6_dry:
            printf("\nStep 6 of 24... %s forearm, %s gel,
%s\n",loc,gel,state);
```

```
            printf("Do you wish to exit [0] OR continue [1] OR repeat
normal force printout [2] OR return to test menu [3] ?\n");
            std::cin >> userResponse;
            while ( userResponse != 0 && userResponse!=1 &&
userResponse!=2 && userResponse!=3) std::cin >> userResponse;
            if ( userResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( userResponse==2 ) goto reDoForceDisplay6;
            else if ( userResponse==3 )
            {
                  goto testMenu;
            }
            else if (userResponse != 0 && userResponse != 1 &&
userResponse != 2 && userResponse != 3 )
            {
                  printf(" \n--Incorrect Entry--\n");
                  goto loc6_dry;
            }
            else if ( userResponse==1 )
            {
                  WetStroking ( device_handler, device,
ticks_per_millisecond, time_next, time_step, time_current_seconds,
xStep, noPositionRot, rotAngles, force_zero, positiveLimit,
negativeLimit, WITH_MAGLEV, WITH_TRAJ, comedi_voltage, force_curr,
desNormForce, thetaY, nextStep, forceActualFiltered,
forceActualCorrected, subjectNum, state, gel, loc, record_DataWet,
test, stepNum, overwriteFilenames );
            }

            printf("\nDo you wish to [0] exit OR [1] continue to Step 7
( Location 1 - Rinse ) OR [2] return to Test Menu ?\n");
            std::cin >> continueResponse;
            if ( continueResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( continueResponse == 2 ) goto testMenu;

      case 7:// Left Distal Rinse 1,4

            stepNum = 7;

            currLoc = 1;
            stateNum = 4;
            gel = showerGelList[currGelOrder[currLoc-1]-1];// first gel
            loc = testLocationList[currLoc-1];// first test location
            state = moistureStateList[stateNum-1];// first state =
initial dry state

            printf("\nStep %i: %s - Location
%i\n",stepNum,state,currLoc);


            if ( detailed_instructions_and_timing )
            {
                  printf("(a) Wring out the clean soaked cloth used
before on Location %d, and fold it in half twice to prepare to remove
the gel/foam from the arm\n",currLoc);
```

```
                printf("(b) While Test Administrator cleans tactor,
Test Subject: wipe the test location only THREE times with the damp
cloth to remove the gel/foam. Unfold/refold the damp cloth as needed to
expose a clean side of the damp cloth for each pass\n");
                printf("(c) Gently dab dry with a clean dry
cloth\n");
                printf("(d) Again place arm face down onto the tactor
with the Location %d oval centered on the tactor, and adjust the
spacers so that the arm is touching lightly\n",currLoc);
                printf("\nSelect [1] to continue...\n");

                std::cin >> userResponse7a;
                while ( userResponse7a != 1 ) std::cin >>
userResponse7a;
          }

          usleep(2000000);
          printf("\nPre force_zero: %f\n",force_zero[2]);
          retractAndZeroForceSensor ( device_handler, device,
cur_pos, nextStep, force_zero, comedi_voltage );
          printf("\nPost force_zero: %f\n",force_zero[2]);

          reDoForceDisplay7:
          forceDisplay ( device, comedi_voltage, force_curr,
ticks_per_millisecond, forceActualFiltered, force_zero, numSeconds );

          // Ask for user response, act accordingly
          loc7_dry:
          printf("\nStep 7 of 24... %s forearm, %s gel,
%s\n",loc,gel,state);
          printf("Do you wish to exit [0] OR continue [1] OR repeat
normal force printout [2] OR return to test menu [3] ?\n");
          std::cin >> userResponse;
          while ( userResponse != 0 && userResponse!=1 &&
userResponse!=2 && userResponse!=3) std::cin >> userResponse;
          if ( userResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
          else if ( userResponse==2 ) goto reDoForceDisplay7;
          else if ( userResponse==3 )
          {
                goto testMenu;
          }
          else if (userResponse != 0 && userResponse != 1 &&
userResponse != 2 && userResponse != 3 )
          {
                printf(" \n--Incorrect Entry--\n");
                goto loc7_dry;
          }
          else if ( userResponse==1 )
          {
                DryStrokingSkinStretch ( device_handler, device,
cur_pos, cur_force, des_force, ticks_per_millisecond, time_next,
time_step, time_current_seconds, record_Data, record_Data_skinStretch,
xStep, dryStrokingTest, skinStretchTest, noPositionRot, rotAngles,
force_zero, positiveLimit, negativeLimit, WITH_MAGLEV, WITH_TRAJ,
comedi_voltage, force_curr, desNormForce, thetaY, nextStep,
```

```
forceActualFiltered, forceActualCorrected, subjectNum, state, gel, loc,
test, stepNum, overwriteFilenames );
            }

            printf("\nDo you wish to [0] exit OR [1] continue to Step 8
( Location 2 - Gel ) OR [2] return to Test Menu ?\n");
            std::cin >> continueResponse;
            if ( continueResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( continueResponse == 2 ) goto testMenu;

      case 8:// Left Proximal Gel 2,2

            stepNum = 8;

            currLoc = 2;
            stateNum = 2;
            gel = showerGelList[currGelOrder[currLoc-1]-1];// first gel
            loc = testLocationList[currLoc-1];// first test location
            state = moistureStateList[stateNum-1];// first state =
initial dry state

            printf("\nStep %i: %s %s - Location
%i\n",stepNum,gel,state,currLoc);

            if ( detailed_instructions_and_timing )
            {
                  printf("(a) Prepare arm-rest spacers so that tactor
touches lightly on Location %d\n",currLoc);
                  printf("(b) Put a new finger cot onto your right
index finger\n");
                  printf("(c) Retrieve the soaked sponge, and without
wringing it out, place it over the oval at Location %d\n",currLoc);
                  printf("(d) Let it rest on Location %d for a period
of %i seconds...\n\nSelect [1] to begin
timer\n",currLoc,wetPrepTimeSeconds);                std::cin >>
userResponse8a;
                  while ( userResponse8a != 1 ) std::cin >>
userResponse8a;
                  if ( userResponse8a == 1 )
                  {
                        time(&start);
                        time(&end);
                        wetPrepTimer = difftime(end,start);
                        //printf("wetPrepTimer = %d",wetPrepTimer);
                        while ( wetPrepTimer < wetPrepTimeSeconds )
                        {
                              time(&end);
                              wetPrepTimer = difftime(end,start);
                              if ( wetPrepTimer != wetPrepTimerOld )
                              {
                                    printf("%i seconds
remaining...\n",(wetPrepTimeSeconds-wetPrepTimer));
                                    wetPrepTimerOld = wetPrepTimer;
                              }
                        }
```

```
                        printf("\n(e) Remove cloth from arm, and place
it back into the clean container\n");
                }

                printf("(f) Apply 1 mL of '%s' shower gel to the oval
at Location %d\n",gel,currLoc);
                printf("(g) Ensure shower gel covers the entire oval
spread it around gently if needed\n");
                printf("(h) Place arm face down onto the tactor with
the oval centered on the tactor, and adjust the spacers if needed\n");
                printf("(i) Keep the finger cot on your finger for
now\n\nSelect [1] to continue...\n");

                std::cin >> userResponse8b;
                while ( userResponse8b != 1 ) std::cin >>
userResponse8b;
          }

          usleep(2000000);
          printf("\nPre force_zero: %f\n",force_zero[2]);
          retractAndZeroForceSensor ( device_handler, device,
cur_pos, nextStep, force_zero, comedi_voltage );
          printf("\nPost force_zero: %f\n",force_zero[2]);

          reDoForceDisplay8:
          forceDisplay ( device, comedi_voltage, force_curr,
ticks_per_millisecond, forceActualFiltered, force_zero, numSeconds );

          // Ask for user response, act accordingly
          loc8_dry:
          printf("\nStep 8 of 24... %s forearm, %s gel,
%s\n",loc,gel,state);
          printf("Do you wish to exit [0] OR continue [1] OR repeat
normal force printout [2] OR return to test menu [3] ?\n");
          std::cin >> userResponse;
          while ( userResponse != 0 && userResponse!=1 &&
userResponse!=2 && userResponse!=3) std::cin >> userResponse;
          if ( userResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
          else if ( userResponse==2 ) goto reDoForceDisplay8;
          else if ( userResponse==3 )
          {
                goto testMenu;
          }
          else if (userResponse != 0 && userResponse != 1 &&
userResponse != 2 && userResponse != 3 )
          {
                printf(" \n--Incorrect Entry--\n");
                goto loc8_dry;
          }
          else if ( userResponse==1 )
          {
                WetStroking ( device_handler, device,
ticks_per_millisecond, time_next, time_step, time_current_seconds,
xStep, noPositionRot, rotAngles, force_zero, positiveLimit,
negativeLimit, WITH_MAGLEV, WITH_TRAJ, comedi_voltage, force_curr,
desNormForce, thetaY, nextStep, forceActualFiltered,
```

```
forceActualCorrected, subjectNum, state, gel, loc, record_DataWet,
test, stepNum, overwriteFilenames );
            }

            printf("\nDo you wish to [0] exit OR [1] continue to Step 9
( Location 2 - Foam ) OR [2] return to Test Menu ?\n");
            std::cin >> continueResponse;
            if ( continueResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( continueResponse == 2 ) goto testMenu;

      case 9:// Left Proximal Foam 2,3

            stepNum = 9;

            currLoc = 2;
            stateNum = 3;
            gel = showerGelList[currGelOrder[currLoc-1]-1];// first gel
            loc = testLocationList[currLoc-1];// first test location
            state = moistureStateList[stateNum-1];// first state =
initial dry state

            printf("\nStep %i: %s %s - Location
%i\n",stepNum,gel,state,currLoc);

            if ( detailed_instructions_and_timing )
            {
                  printf("(a) Get ready to lather the shower gel at
Location %d by rubbing vigorously for %d
seconds\n",currLoc,foamPrepTimeSeconds);
                  printf("(b) Select [1] to start the %d second
timer.\nTest Subject: begin lathering\nTest Administrator: clean
tactor\n",foamPrepTimeSeconds);
                  std::cin >> userResponse9a;
                  while ( userResponse9a != 1 ) std::cin >>
userResponse9a;
                  if ( userResponse9a == 1 )
                  {
                        time(&start);
                        time(&end);
                        foamPrepTimer = difftime(end,start);
                        //printf("wetPrepTimer = %d",wetPrepTimer);
                        while ( foamPrepTimer < foamPrepTimeSeconds )
                        {
                              time(&end);
                              foamPrepTimer = difftime(end,start);
                              if ( foamPrepTimer != foamPrepTimerOld )
                              {
                                    printf("%i seconds
remaining...\n",(foamPrepTimeSeconds-foamPrepTimer));
                                    foamPrepTimerOld = foamPrepTimer;
                              }
                        }
                  }
                  //else if ( userResponse == 2 ) goto TestMenu;
                  printf("(c) Remove finger cot\n");
```

```
                  printf("(d) Again place arm face down onto the tactor
with the Location %d oval centered on the tactor, and adjust the
spacers so that the arm is touching lightly\n",currLoc);
                  printf("\nSelect [1] to continue...\n");

                  std::cin >> userResponse9b;
                  while ( userResponse9b != 1 ) std::cin >>
userResponse9b;
            }

            usleep(2000000);
            printf("\nPre force_zero: %f\n",force_zero[2]);
            retractAndZeroForceSensor ( device_handler, device,
cur_pos, nextStep, force_zero, comedi_voltage );
            printf("\nPost force_zero: %f\n",force_zero[2]);

            reDoForceDisplay9:
            forceDisplay ( device, comedi_voltage, force_curr,
ticks_per_millisecond, forceActualFiltered, force_zero, numSeconds );

            // Ask for user response, act accordingly
            loc9_dry:
            printf("\nStep 9 of 24... %s forearm, %s gel,
%s\n",loc,gel,state);
            printf("Do you wish to exit [0] OR continue [1] OR repeat
normal force printout [2] OR return to test menu [3] ?\n");
            std::cin >> userResponse;
            while ( userResponse != 0 && userResponse!=1 &&
userResponse!=2 && userResponse!=3) std::cin >> userResponse;
            if ( userResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( userResponse==2 ) goto reDoForceDisplay9;
            else if ( userResponse==3 )
            {
                  goto testMenu;
            }
            else if (userResponse != 0 && userResponse != 1 &&
userResponse != 2 && userResponse != 3 )
            {
                  printf(" \n--Incorrect Entry--\n");
                  goto loc9_dry;
            }
            else if ( userResponse==1 )
            {
                  WetStroking ( device_handler, device,
ticks_per_millisecond, time_next, time_step, time_current_seconds,
xStep, noPositionRot, rotAngles, force_zero, positiveLimit,
negativeLimit, WITH_MAGLEV, WITH_TRAJ, comedi_voltage, force_curr,
desNormForce, thetaY, nextStep, forceActualFiltered,
forceActualCorrected, subjectNum, state, gel, loc, record_DataWet,
test, stepNum, overwriteFilenames );
            }

            printf("\nDo you wish to [0] exit OR [1] continue to Step
10 ( Location 2 - Rinse ) OR [2] return to Test Menu ?\n");
            std::cin >> continueResponse;
```

```
            if ( continueResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( continueResponse == 2 ) goto testMenu;

      case 10:// Left Proximal Rinse 2,4

            stepNum = 10;

            currLoc = 2;
            stateNum = 4;
            gel = showerGelList[currGelOrder[currLoc-1]-1];// first gel
            loc = testLocationList[currLoc-1];// first test location
            state = moistureStateList[stateNum-1];// first state =
initial dry state

            printf("\nStep %i: %s - Location
%i\n",stepNum,state,currLoc);


            if ( detailed_instructions_and_timing )
            {
                  printf("(a) Wring out the clean soaked cloth used
before on Location %d, and fold it in half twice to prepare to remove
the gel/foam from the arm\n",currLoc);
                  printf("(b) While Test Administrator cleans tactor,
Test Subject: wipe the test location only THREE times with the damp
cloth to remove the gel/foam. Unfold/refold the damp cloth as needed to
expose a clean side of the damp cloth for each pass\n");
                  printf("(c) Gently dab dry with a clean dry
cloth\n");
                  printf("(d) Again place arm face down onto the tactor
with the Location %d oval centered on the tactor, and adjust the
spacers so that the arm is touching lightly\n",currLoc);
                  printf("\nSelect [1] to continue...\n");

                  std::cin >> userResponse10a;
                  while ( userResponse10a != 1 ) std::cin >>
userResponse10a;
            }

            usleep(2000000);
            printf("\nPre force_zero: %f\n",force_zero[2]);
            retractAndZeroForceSensor ( device_handler, device,
cur_pos, nextStep, force_zero, comedi_voltage );
            printf("\nPost force_zero: %f\n",force_zero[2]);

            reDoForceDisplay10:
            forceDisplay ( device, comedi_voltage, force_curr,
ticks_per_millisecond, forceActualFiltered, force_zero, numSeconds );

            // Ask for user response, act accordingly
            loc10_dry:
            printf("\nStep 10 of 24... %s forearm, %s gel,
%s\n",loc,gel,state);
            printf("Do you wish to exit [0] OR continue [1] OR repeat
normal force printout [2] OR return to test menu [3] ?\n");
            std::cin >> userResponse;
```

```
            while ( userResponse != 0 && userResponse!=1 &&
userResponse!=2 && userResponse!=3) std::cin >> userResponse;
            if ( userResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( userResponse==2 ) goto reDoForceDisplay10;
            else if ( userResponse==3 )
            {
                  goto testMenu;
            }
            else if (userResponse != 0 && userResponse != 1 &&
userResponse != 2 && userResponse != 3 )
            {
                  printf(" \n--Incorrect Entry--\n");
                  goto loc10_dry;
            }
            else if ( userResponse==1 )
            {
                  DryStrokingSkinStretch ( device_handler, device,
cur_pos, cur_force, des_force, ticks_per_millisecond, time_next,
time_step, time_current_seconds, record_Data, record_Data_skinStretch,
xStep, dryStrokingTest, skinStretchTest, noPositionRot, rotAngles,
force_zero, positiveLimit, negativeLimit, WITH_MAGLEV, WITH_TRAJ,
comedi_voltage, force_curr, desNormForce, thetaY, nextStep,
forceActualFiltered, forceActualCorrected, subjectNum, state, gel, loc,
test, stepNum, overwriteFilenames );
            }

            printf("\nDo you wish to [0] exit OR [1] continue to Step
11 ( Location 1 - Wait to ReDry ) OR [2] return to Test Menu ?\n");
            std::cin >> continueResponse;
            if ( continueResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( continueResponse == 2 ) goto testMenu;

      case 11:// Left Distal WAIT

            printf("Get ready to test Location 1 for the last time, but
please WAIT to begin...");
            stepNum = 11;

            currLoc = 1;
            loc = testLocationList[currLoc-1];

            secondsAtRinse = getSecondsAtRinse ( loc, year, month,
date, subjectNum, test );

            while ( secondsSinceRinse < goalTime )
            {
                  getDateTimeStr ( year, month, date, hours, minutes,
seconds );
                  secondsCurrent = (atoi(hours)*3600 + atoi(minutes)*60
+ atoi(seconds));

                  secondsSinceRinse = secondsCurrent - secondsAtRinse;
                  waitTime_loc1 = goalTime - (double)secondsSinceRinse;

                  waitTimeSeconds_loc1 = int(waitTime_loc1) % 60;
```

```
                  waitTime_loc1 = waitTime_loc1/60.0;
                  waitTimeInt_loc1 = (int)waitTime_loc1;
                  if ( waitTimeOld_loc1 != waitTime_loc1 )
                  {
                        printf("\nPlease wait to begin Step
12\tLocation 1 - ReDry\t%02d:%02d
",waitTimeInt_loc1,waitTimeSeconds_loc1);
                        waitTimeOld_loc1=waitTime_loc1;
                  }
            }

            secondsAtRinse = 0;
            secondsSinceRinse = 0;
            secondsCurrent = 0;
            minutesSinceRinse = 0;

            printf("\nThe 20 minute wait is complete.");

            printf("\nDo you wish to [0] exit OR [1] continue to Step
12 ( Location 1 - ReDry ) OR [2] return to Test Menu ?\n");
            std::cin >> continueResponse;
            if ( continueResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( continueResponse == 2 ) goto testMenu;

      case 12:// Left Distal ReDry 1,5

            stepNum = 12;

            currLoc = 1;
            stateNum = 5;
            gel = showerGelList[currGelOrder[currLoc-1]-1];// first gel
            loc = testLocationList[currLoc-1];// first test location
            state = moistureStateList[stateNum-1];// first state =
initial dry state

            printf("\nStep %i: %s - Location
%i\n",stepNum,state,currLoc);


            usleep(2000000);
            printf("\nPre force_zero: %f\n",force_zero[2]);
            retractAndZeroForceSensor ( device_handler, device,
cur_pos, nextStep, force_zero, comedi_voltage );
            printf("\nPost force_zero: %f\n",force_zero[2]);

            reDoForceDisplay12:
            forceDisplay ( device, comedi_voltage, force_curr,
ticks_per_millisecond, forceActualFiltered, force_zero, numSeconds );

            // Ask for user response, act accordingly
            loc12_dry:
            printf("\nStep 12 of 24... %s forearm, %s gel,
%s\n",loc,gel,state);
            printf("Do you wish to exit [0] OR continue [1] OR repeat
normal force printout [2] OR return to test menu [3] ?\n");
            std::cin >> userResponse;
```

```
            while ( userResponse != 0 && userResponse!=1 &&
userResponse!=2 && userResponse!=3) std::cin >> userResponse;
            if ( userResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( userResponse==2 ) goto reDoForceDisplay12;
            else if ( userResponse==3 )
            {
                  goto testMenu;
            }
            else if (userResponse != 0 && userResponse != 1 &&
userResponse != 2 && userResponse != 3 )
            {
                  printf(" \n--Incorrect Entry--\n");
                  goto loc12_dry;
            }
            else if ( userResponse==1 )
            {
                  DryStrokingSkinStretch ( device_handler, device,
cur_pos, cur_force, des_force, ticks_per_millisecond, time_next,
time_step, time_current_seconds, record_Data, record_Data_skinStretch,
xStep, dryStrokingTest, skinStretchTest, noPositionRot, rotAngles,
force_zero, positiveLimit, negativeLimit, WITH_MAGLEV, WITH_TRAJ,
comedi_voltage, force_curr, desNormForce, thetaY, nextStep,
forceActualFiltered, forceActualCorrected, subjectNum, state, gel, loc,
test, stepNum, overwriteFilenames );
            }

            printf("\nDo you wish to [0] exit OR [1] continue to Step
13 ( Location 3 - Gel ) OR [2] return to Test Menu ?\n");
            std::cin >> continueResponse;
            if ( continueResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( continueResponse == 2 ) goto testMenu;

      case 13:// Right Distal Gel 3,2

            stepNum = 13;

            currLoc = 3;
            stateNum = 2;
            gel = showerGelList[currGelOrder[currLoc-1]-1];// first gel
            loc = testLocationList[currLoc-1];// first test location
            state = moistureStateList[stateNum-1];// first state =
initial dry state

            printf("\nStep %i: %s %s - Location
%i\n",stepNum,gel,state,currLoc);

            if ( detailed_instructions_and_timing )
            {
                  printf("(a) Prepare arm-rest spacers so that tactor
touches lightly on Location %d\n",currLoc);
                  printf("(b) Put a new finger cot onto your left index
finger\n");
                  printf("(c) Retrieve the soaked sponge, and without
wringing it out, place it over the oval at Location %d\n",currLoc);
```

```
                  printf("(d) Let it rest on Location %d for a period
of %i seconds...\n\nSelect [1] to begin
timer\n",currLoc,wetPrepTimeSeconds);           std::cin >>
userResponse13a;
                  while ( userResponse13a != 1 ) std::cin >>
userResponse13a;
                  if ( userResponse13a == 1 )
                  {
                        time(&start);
                        time(&end);
                        wetPrepTimer = difftime(end,start);
                        //printf("wetPrepTimer = %d",wetPrepTimer);
                        while ( wetPrepTimer < wetPrepTimeSeconds )
                        {
                              time(&end);
                              wetPrepTimer = difftime(end,start);
                              if ( wetPrepTimer != wetPrepTimerOld )
                              {
                                    printf("%i seconds
remaining...\n",(wetPrepTimeSeconds-wetPrepTimer));
                                    wetPrepTimerOld = wetPrepTimer;
                              }
                        }
                        printf("\n(e) Remove cloth from arm, and place
it back into the clean container\n");
                  }
                  //else if ( userResponse == 2 ) goto TestMenu;

                  printf("(f) Apply 1 mL of '%s' shower gel to the oval
at Location %d\n",gel,currLoc);
                  printf("(g) Ensure shower gel covers the entire oval
spread it around gently if needed\n");
                  printf("(h) Place arm face down onto the tactor with
the oval centered on the tactor, and adjust the spacers if needed\n");
                  printf("(i) Keep the finger cot on your finger for
now\n\nSelect [1] to continue...\n");

                  std::cin >> userResponse13b;
                  while ( userResponse13b != 1 ) std::cin >>
userResponse13b;
            }

            usleep(2000000);
            printf("\nPre force_zero: %f\n",force_zero[2]);
            retractAndZeroForceSensor ( device_handler, device,
cur_pos, nextStep, force_zero, comedi_voltage );
            printf("\nPost force_zero: %f\n",force_zero[2]);

            reDoForceDisplay13:
            forceDisplay ( device, comedi_voltage, force_curr,
ticks_per_millisecond, forceActualFiltered, force_zero, numSeconds );

            // Ask for user response, act accordingly
            loc13_dry:
            printf("\nStep 13 of 24... %s forearm, %s gel,
%s\n",loc,gel,state);
```

```
            printf("Do you wish to exit [0] OR continue [1] OR repeat
normal force printout [2] OR return to test menu [3] ?\n");
            std::cin >> userResponse;
            while ( userResponse != 0 && userResponse!=1 &&
userResponse!=2 && userResponse!=3) std::cin >> userResponse;
            if ( userResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( userResponse==2 ) goto reDoForceDisplay13;
            else if ( userResponse==3 )
            {
                  goto testMenu;
            }
            else if (userResponse != 0 && userResponse != 1 &&
userResponse != 2 && userResponse != 3 )
            {
                  printf(" \n--Incorrect Entry--\n");
                  goto loc13_dry;
            }
            else if ( userResponse==1 )
            {
                  WetStroking ( device_handler, device,
ticks_per_millisecond, time_next, time_step, time_current_seconds,
xStep, noPositionRot, rotAngles, force_zero, positiveLimit,
negativeLimit, WITH_MAGLEV, WITH_TRAJ, comedi_voltage, force_curr,
desNormForce, thetaY, nextStep, forceActualFiltered,
forceActualCorrected, subjectNum, state, gel, loc, record_DataWet,
test, stepNum, overwriteFilenames );
            }

            printf("\nDo you wish to [0] exit OR [1] continue to Step
14 ( Location 3 - Foam ) OR [2] return to Test Menu ?\n");
            std::cin >> continueResponse;
            if ( continueResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( continueResponse == 2 ) goto testMenu;

      case 14:// Right Distal Foam 3,3

            stepNum = 14;

            currLoc = 3;
            stateNum = 3;
            gel = showerGelList[currGelOrder[currLoc-1]-1];// first gel
            loc = testLocationList[currLoc-1];// first test location
            state = moistureStateList[stateNum-1];// first state =
initial dry state

            printf("\nStep %i: %s %s - Location
%i\n",stepNum,gel,state,currLoc);

            if ( detailed_instructions_and_timing )
            {
                  printf("(a) Get ready to lather the shower gel at
Location %d by rubbing vigorously for %d
seconds\n",currLoc,foamPrepTimeSeconds);
```

```
                printf("(b) Select [1] to start the %d second
timer.\nTest Subject: begin lathering\nTest Administrator: clean
tactor\n",foamPrepTimeSeconds);
                std::cin >> userResponse14a;
                while ( userResponse14a != 1 ) std::cin >>
userResponse14a;
                if ( userResponse14a == 1 )
                {
                      time(&start);
                      time(&end);
                      foamPrepTimer = difftime(end,start);
                      //printf("wetPrepTimer = %d",wetPrepTimer);
                      while ( foamPrepTimer < foamPrepTimeSeconds )
                      {
                            time(&end);
                            foamPrepTimer = difftime(end,start);
                            if ( foamPrepTimer != foamPrepTimerOld )
                            {
                                  printf("%i seconds
remaining...\n",(foamPrepTimeSeconds-foamPrepTimer));
                                  foamPrepTimerOld = foamPrepTimer;
                            }
                      }
                }
                //else if ( userResponse == 2 ) goto TestMenu;
                printf("(c) Remove finger cot\n");
                printf("(d) Again place arm face down onto the tactor
with the Location %d oval centered on the tactor, and adjust the
spacers so that the arm is touching lightly\n",currLoc);
                printf("\nSelect [1] to continue...\n");

                std::cin >> userResponse14b;
                while ( userResponse14b != 1 ) std::cin >>
userResponse14b;
            }

            usleep(2000000);
            printf("\nPre force_zero: %f\n",force_zero[2]);
            retractAndZeroForceSensor ( device_handler, device,
cur_pos, nextStep, force_zero, comedi_voltage );
            printf("\nPost force_zero: %f\n",force_zero[2]);

            reDoForceDisplay14:
            forceDisplay ( device, comedi_voltage, force_curr,
ticks_per_millisecond, forceActualFiltered, force_zero, numSeconds );

            // Ask for user response, act accordingly
            loc14_dry:
            printf("\nStep 14 of 24... %s forearm, %s gel,
%s\n",loc,gel,state);
            printf("Do you wish to exit [0] OR continue [1] OR repeat
normal force printout [2] OR return to test menu [3] ?\n");
            std::cin >> userResponse;
            while ( userResponse != 0 && userResponse!=1 &&
userResponse!=2 && userResponse!=3) std::cin >> userResponse;
            if ( userResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
```

```
            else if ( userResponse==2 ) goto reDoForceDisplay14;
            else if ( userResponse==3 )
            {
                  goto testMenu;
            }
            else if (userResponse != 0 && userResponse != 1 &&
userResponse != 2 && userResponse != 3 )
            {
                  printf(" \n--Incorrect Entry--\n");
                  goto loc14_dry;
            }
            else if ( userResponse==1 )
            {
                  WetStroking ( device_handler, device,
ticks_per_millisecond, time_next, time_step, time_current_seconds,
xStep, noPositionRot, rotAngles, force_zero, positiveLimit,
negativeLimit, WITH_MAGLEV, WITH_TRAJ, comedi_voltage, force_curr,
desNormForce, thetaY, nextStep, forceActualFiltered,
forceActualCorrected, subjectNum, state, gel, loc, record_DataWet,
test, stepNum, overwriteFilenames );
            }

            printf("\nDo you wish to [0] exit OR [1] continue to Step
15 ( Location 3 - Rinse ) OR [2] return to Test Menu ?\n");
            std::cin >> continueResponse;
            if ( continueResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( continueResponse == 2 ) goto testMenu;

      case 15:// Right Distal Rinse 3,4

            stepNum = 15;

            currLoc = 3;
            stateNum = 4;
            gel = showerGelList[currGelOrder[currLoc-1]-1];// first gel
            loc = testLocationList[currLoc-1];// first test location
            state = moistureStateList[stateNum-1];// first state =
initial dry state

            printf("\nStep %i: %s - Location
%i\n",stepNum,state,currLoc);


            if ( detailed_instructions_and_timing )
            {
                  printf("(a) Wring out the clean soaked cloth used
before on Location %d, and fold it in half twice to prepare to remove
the gel/foam from the arm\n",currLoc);
                  printf("(b) While Test Administrator cleans tactor,
Test Subject: wipe the test location only THREE times with the damp
cloth to remove the gel/foam. Unfold/refold the damp cloth as needed to
expose a clean side of the damp cloth for each pass\n");
                  printf("(c) Gently dab dry with a clean dry
cloth\n");
```

```
                  printf("(d) Again place arm face down onto the tactor
with the Location %d oval centered on the tactor, and adjust the
spacers so that the arm is touching lightly\n",currLoc);
                  printf("\nSelect [1] to continue...\n");

                  std::cin >> userResponse15a;
                  while ( userResponse15a != 1 ) std::cin >>
userResponse15a;
            }

            usleep(2000000);
            printf("\nPre force_zero: %f\n",force_zero[2]);
            retractAndZeroForceSensor ( device_handler, device,
cur_pos, nextStep, force_zero, comedi_voltage );
            printf("\nPost force_zero: %f\n",force_zero[2]);

            reDoForceDisplay15:
            forceDisplay ( device, comedi_voltage, force_curr,
ticks_per_millisecond, forceActualFiltered, force_zero, numSeconds );

            // Ask for user response, act accordingly
            loc15_dry:
            printf("\nStep 15 of 24... %s forearm, %s gel,
%s\n",loc,gel,state);
            printf("Do you wish to exit [0] OR continue [1] OR repeat
normal force printout [2] OR return to test menu [3] ?\n");
            std::cin >> userResponse;
            while ( userResponse != 0 && userResponse!=1 &&
userResponse!=2 && userResponse!=3) std::cin >> userResponse;
            if ( userResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( userResponse==2 ) goto reDoForceDisplay15;
            else if ( userResponse==3 )
            {
                  goto testMenu;
            }
            else if (userResponse != 0 && userResponse != 1 &&
userResponse != 2 && userResponse != 3 )
            {
                  printf(" \n--Incorrect Entry--\n");
                  goto loc15_dry;
            }
            else if ( userResponse==1 )
            {
                  DryStrokingSkinStretch ( device_handler, device,
cur_pos, cur_force, des_force, ticks_per_millisecond, time_next,
time_step, time_current_seconds, record_Data, record_Data_skinStretch,
xStep, dryStrokingTest, skinStretchTest, noPositionRot, rotAngles,
force_zero, positiveLimit, negativeLimit, WITH_MAGLEV, WITH_TRAJ,
comedi_voltage, force_curr, desNormForce, thetaY, nextStep,
forceActualFiltered, forceActualCorrected, subjectNum, state, gel, loc,
test, stepNum, overwriteFilenames );
            }

            printf("\nDo you wish to [0] exit OR [1] continue to Step
16 ( Location 2 - Wait for ReDry ) OR [2] return to Test Menu ?\n");
            std::cin >> continueResponse;
```

```
            if ( continueResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( continueResponse == 2 ) goto testMenu;

      case 16:// Left Proximal WAIT

            printf("Get ready to test Location 2 for the last time, but
please WAIT to begin...");
            stepNum = 16;

            currLoc = 2;
            loc = testLocationList[currLoc-1];

            secondsAtRinse = getSecondsAtRinse ( loc, year, month,
date, subjectNum, test );

            while ( secondsSinceRinse < goalTime )
            {
                  getDateTimeStr ( year, month, date, hours, minutes,
seconds );
                  secondsCurrent = (atoi(hours)*3600 + atoi(minutes)*60
+ atoi(seconds));

                  secondsSinceRinse = secondsCurrent - secondsAtRinse;
                  waitTime_loc1 = goalTime - (double)secondsSinceRinse;

                  waitTimeSeconds_loc1 = int(waitTime_loc1) % 60;
                  waitTime_loc1 = waitTime_loc1/60.0;
                  waitTimeInt_loc1 = (int)waitTime_loc1;
                  if ( waitTimeOld_loc1 != waitTime_loc1 )
                  {
                        printf("\nPlease wait to begin Step
17\tLocation 2 - ReDry\t%02d:%02d
",waitTimeInt_loc1,waitTimeSeconds_loc1);
                        waitTimeOld_loc1=waitTime_loc1;
                  }
            }

            secondsAtRinse = 0;
            secondsSinceRinse = 0;
            secondsCurrent = 0;
            minutesSinceRinse = 0;

            printf("\nThe 20 minute wait is complete.");

            printf("\nDo you wish to [0] exit OR [1] continue to Step
17 ( Location 2 - ReDry ) OR [2] return to Test Menu ?\n");
            std::cin >> continueResponse;
            if ( continueResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( continueResponse == 2 ) goto testMenu;

      case 17:// Left Proximal ReDry 2,5

            stepNum = 17;

            currLoc = 2;
```

```
            stateNum = 5;
            gel = showerGelList[currGelOrder[currLoc-1]-1];// first gel
            loc = testLocationList[currLoc-1];// first test location
            state = moistureStateList[stateNum-1];// first state =
initial dry state

            printf("\nStep %i: %s - Location
%i\n",stepNum,state,currLoc);


            usleep(2000000);
            printf("\nPre force_zero: %f\n",force_zero[2]);
            retractAndZeroForceSensor ( device_handler, device,
cur_pos, nextStep, force_zero, comedi_voltage );
            printf("\nPost force_zero: %f\n",force_zero[2]);

            reDoForceDisplay17:
            forceDisplay ( device, comedi_voltage, force_curr,
ticks_per_millisecond, forceActualFiltered, force_zero, numSeconds );

            // Ask for user response, act accordingly
            loc17_dry:
            printf("\nStep 17 of 24... %s forearm, %s gel,
%s\n",loc,gel,state);
            printf("Do you wish to exit [0] OR continue [1] OR repeat
normal force printout [2] OR return to test menu [3] ?\n");
            std::cin >> userResponse;
            while ( userResponse != 0 && userResponse!=1 &&
userResponse!=2 && userResponse!=3) std::cin >> userResponse;
            if ( userResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( userResponse==2 ) goto reDoForceDisplay17;
            else if ( userResponse==3 )
            {
                  goto testMenu;
            }
            else if (userResponse != 0 && userResponse != 1 &&
userResponse != 2 && userResponse != 3 )
            {
                  printf(" \n--Incorrect Entry--\n");
                  goto loc17_dry;
            }
            else if ( userResponse==1 )
            {
                  DryStrokingSkinStretch ( device_handler, device,
cur_pos, cur_force, des_force, ticks_per_millisecond, time_next,
time_step, time_current_seconds, record_Data, record_Data_skinStretch,
xStep, dryStrokingTest, skinStretchTest, noPositionRot, rotAngles,
force_zero, positiveLimit, negativeLimit, WITH_MAGLEV, WITH_TRAJ,
comedi_voltage, force_curr, desNormForce, thetaY, nextStep,
forceActualFiltered, forceActualCorrected, subjectNum, state, gel, loc,
test, stepNum, overwriteFilenames );
            }

            printf("\nDo you wish to [0] exit OR [1] continue to Step
18 ( Location 4 - Gel ) OR [2] return to Test Menu ?\n");
            std::cin >> continueResponse;
```

```
            if ( continueResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( continueResponse == 2 ) goto testMenu;


      case 18:// Right Proximal Gel 4,2

            stepNum = 18;

            currLoc = 4;
            stateNum = 2;
            gel = showerGelList[currGelOrder[currLoc-1]-1];// first gel
            loc = testLocationList[currLoc-1];// first test location
            state = moistureStateList[stateNum-1];// first state =
initial dry state

            printf("\nStep %i: %s %s - Location
%i\n",stepNum,gel,state,currLoc);

            if ( detailed_instructions_and_timing )
            {
                  printf("(a) Prepare arm-rest spacers so that tactor
touches lightly on Location %d\n",currLoc);
                  printf("(b) Put a new finger cot onto your left index
finger\n");
                  printf("(c) Retrieve the soaked sponge, and without
wringing it out, place it over the oval at Location %d\n",currLoc);
                  printf("(d) Let it rest on Location %d for a period
of %i seconds...\n\nSelect [1] to begin
timer\n",currLoc,wetPrepTimeSeconds);           std::cin >>
userResponse18a;
                  while ( userResponse18a != 1 ) std::cin >>
userResponse18a;
                  if ( userResponse18a == 1 )
                  {
                        time(&start);
                        time(&end);
                        wetPrepTimer = difftime(end,start);
                        //printf("wetPrepTimer = %d",wetPrepTimer);
                        while ( wetPrepTimer < wetPrepTimeSeconds )
                        {
                              time(&end);
                              wetPrepTimer = difftime(end,start);
                              if ( wetPrepTimer != wetPrepTimerOld )
                              {
                                    printf("%i seconds
remaining...\n",(wetPrepTimeSeconds-wetPrepTimer));
                                    wetPrepTimerOld = wetPrepTimer;
                              }
                        }
                        printf("\n(e) Remove cloth from arm, and place
it back into the clean container\n");
                  }
                  //else if ( userResponse == 2 ) goto TestMenu;

                  printf("(f) Apply 1 mL of '%s' shower gel to the oval
at Location %d\n",gel,currLoc);
```

```
                  printf("(g) Ensure shower gel covers the entire oval
spread it around gently if needed\n");
                  printf("(h) Place arm face down onto the tactor with
the oval centered on the tactor, and adjust the spacers if needed\n");
                  printf("(i) Keep the finger cot on your finger for
now\n\nSelect [1] to continue...\n");

                  std::cin >> userResponse18b;
                  while ( userResponse18b != 1 ) std::cin >>
userResponse18b;
            }

            usleep(2000000);
            printf("\nPre force_zero: %f\n",force_zero[2]);
            retractAndZeroForceSensor ( device_handler, device,
cur_pos, nextStep, force_zero, comedi_voltage );
            printf("\nPost force_zero: %f\n",force_zero[2]);

            reDoForceDisplay18:
            forceDisplay ( device, comedi_voltage, force_curr,
ticks_per_millisecond, forceActualFiltered, force_zero, numSeconds );

            // Ask for user response, act accordingly
            loc18_dry:
            printf("\nStep 18 of 24... %s forearm, %s gel,
%s\n",loc,gel,state);
            printf("Do you wish to exit [0] OR continue [1] OR repeat
normal force printout [2] OR return to test menu [3] ?\n");
            std::cin >> userResponse;
            while ( userResponse != 0 && userResponse!=1 &&
userResponse!=2 && userResponse!=3) std::cin >> userResponse;
            if ( userResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( userResponse==2 ) goto reDoForceDisplay18;
            else if ( userResponse==3 )
            {
                  goto testMenu;
            }
            else if (userResponse != 0 && userResponse != 1 &&
userResponse != 2 && userResponse != 3 )
            {
                  printf(" \n--Incorrect Entry--\n");
                  goto loc18_dry;
            }
            else if ( userResponse==1 )
            {
                  WetStroking ( device_handler, device,
ticks_per_millisecond, time_next, time_step, time_current_seconds,
xStep, noPositionRot, rotAngles, force_zero, positiveLimit,
negativeLimit, WITH_MAGLEV, WITH_TRAJ, comedi_voltage, force_curr,
desNormForce, thetaY, nextStep, forceActualFiltered,
forceActualCorrected, subjectNum, state, gel, loc, record_DataWet,
test, stepNum, overwriteFilenames );
            }

            printf("\nDo you wish to [0] exit OR [1] continue to Step
19 ( Location 4 - Foam ) OR [2] return to Test Menu ?\n");
```

```
            std::cin >> continueResponse;
            if ( continueResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( continueResponse == 2 ) goto testMenu;

      case 19:// Right Proximal Foam 4,3

            stepNum = 19;

            currLoc = 4;
            stateNum = 3;
            gel = showerGelList[currGelOrder[currLoc-1]-1];// first gel
            loc = testLocationList[currLoc-1];// first test location
            state = moistureStateList[stateNum-1];// first state =
initial dry state

            printf("\nStep %i: %s %s - Location
%i\n",stepNum,gel,state,currLoc);

            if ( detailed_instructions_and_timing )
            {
                  printf("(a) Get ready to lather the shower gel at
Location %d by rubbing vigorously for %d
seconds\n",currLoc,foamPrepTimeSeconds);
                  printf("(b) Select [1] to start the %d second
timer.\nTest Subject: begin lathering\nTest Administrator: clean
tactor\n",foamPrepTimeSeconds);
                  std::cin >> userResponse19a;
                  while ( userResponse19a != 1 ) std::cin >>
userResponse19a;
                  if ( userResponse19a == 1 )
                  {
                        time(&start);
                        time(&end);
                        foamPrepTimer = difftime(end,start);
                        //printf("wetPrepTimer = %d",wetPrepTimer);
                        while ( foamPrepTimer < foamPrepTimeSeconds )
                        {
                              time(&end);
                              foamPrepTimer = difftime(end,start);
                              if ( foamPrepTimer != foamPrepTimerOld )
                              {
                                    printf("%i seconds
remaining...\n",(foamPrepTimeSeconds-foamPrepTimer));
                                    foamPrepTimerOld = foamPrepTimer;
                              }
                        }
                  }
                  //else if ( userResponse == 2 ) goto TestMenu;
                  printf("(c) Remove finger cot\n");
                  printf("(d) Again place arm face down onto the tactor
with the Location %d oval centered on the tactor, and adjust the
spacers so that the arm is touching lightly\n",currLoc);
                  printf("\nSelect [1] to continue...\n");

                  std::cin >> userResponse19b;
```

```
                  while ( userResponse19b != 1 ) std::cin >>
userResponse19b;
            }

            usleep(2000000);
            printf("\nPre force_zero: %f\n",force_zero[2]);
            retractAndZeroForceSensor ( device_handler, device,
cur_pos, nextStep, force_zero, comedi_voltage );
            printf("\nPost force_zero: %f\n",force_zero[2]);

            reDoForceDisplay19:
            forceDisplay ( device, comedi_voltage, force_curr,
ticks_per_millisecond, forceActualFiltered, force_zero, numSeconds );

            // Ask for user response, act accordingly
            loc19_dry:
            printf("\nStep 19 of 24... %s forearm, %s gel,
%s\n",loc,gel,state);
            printf("Do you wish to exit [0] OR continue [1] OR repeat
normal force printout [2] OR return to test menu [3] ?\n");
            std::cin >> userResponse;
            while ( userResponse != 0 && userResponse!=1 &&
userResponse!=2 && userResponse!=3) std::cin >> userResponse;
            if ( userResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( userResponse==2 ) goto reDoForceDisplay19;
            else if ( userResponse==3 )
            {
                  goto testMenu;
            }
            else if (userResponse != 0 && userResponse != 1 &&
userResponse != 2 && userResponse != 3 )
            {
                  printf(" \n--Incorrect Entry--\n");
                  goto loc19_dry;
            }
            else if ( userResponse==1 )
            {
                  WetStroking ( device_handler, device,
ticks_per_millisecond, time_next, time_step, time_current_seconds,
xStep, noPositionRot, rotAngles, force_zero, positiveLimit,
negativeLimit, WITH_MAGLEV, WITH_TRAJ, comedi_voltage, force_curr,
desNormForce, thetaY, nextStep, forceActualFiltered,
forceActualCorrected, subjectNum, state, gel, loc, record_DataWet,
test, stepNum, overwriteFilenames );
            }

            printf("\nDo you wish to [0] exit OR [1] continue to Step
20 ( Location 4 - Rinse ) OR [2] return to Test Menu ?\n");
            std::cin >> continueResponse;
            if ( continueResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( continueResponse == 2 ) goto testMenu;

      case 20:// Right Proximal Rinse 4,4

            stepNum = 20;
```

```
            currLoc = 4;
            stateNum = 4;
            gel = showerGelList[currGelOrder[currLoc-1]-1];// first gel
            loc = testLocationList[currLoc-1];// first test location
            state = moistureStateList[stateNum-1];// first state =
initial dry state

            printf("\nStep %i: %s - Location
%i\n",stepNum,state,currLoc);


            if ( detailed_instructions_and_timing )
            {
                  printf("(a) Wring out the clean soaked cloth used
before on Location %d, and fold it in half twice to prepare to remove
the gel/foam from the arm\n",currLoc);
                  printf("(b) While Test Administrator cleans tactor,
Test Subject: wipe the test location only THREE times with the damp
cloth to remove the gel/foam. Unfold/refold the damp cloth as needed to
expose a clean side of the damp cloth for each pass\n");
                  printf("(c) Gently dab dry with a clean dry
cloth\n");
                  printf("(d) Again place arm face down onto the tactor
with the Location %d oval centered on the tactor, and adjust the
spacers so that the arm is touching lightly\n",currLoc);
                  printf("\nSelect [1] to continue...\n");

                  std::cin >> userResponse20a;
                  while ( userResponse20a != 1 ) std::cin >>
userResponse20a;
            }

            usleep(2000000);
            printf("\nPre force_zero: %f\n",force_zero[2]);
            retractAndZeroForceSensor ( device_handler, device,
cur_pos, nextStep, force_zero, comedi_voltage );
            printf("\nPost force_zero: %f\n",force_zero[2]);

            reDoForceDisplay20:
            forceDisplay ( device, comedi_voltage, force_curr,
ticks_per_millisecond, forceActualFiltered, force_zero, numSeconds );

            // Ask for user response, act accordingly
            loc20_dry:
            printf("\nStep 20 of 24... %s forearm, %s gel,
%s\n",loc,gel,state);
            printf("Do you wish to exit [0] OR continue [1] OR repeat
normal force printout [2] OR return to test menu [3] ?\n");
            std::cin >> userResponse;
            while ( userResponse != 0 && userResponse!=1 &&
userResponse!=2 && userResponse!=3) std::cin >> userResponse;
            if ( userResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( userResponse==2 ) goto reDoForceDisplay20;
            else if ( userResponse==3 )
            {
```

```
                goto testMenu;
            }
            else if (userResponse != 0 && userResponse != 1 &&
userResponse != 2 && userResponse != 3 )
            {
                printf(" \n--Incorrect Entry--\n");
                goto loc20_dry;
            }
            else if ( userResponse==1 )
            {
                DryStrokingSkinStretch ( device_handler, device,
cur_pos, cur_force, des_force, ticks_per_millisecond, time_next,
time_step, time_current_seconds, record_Data, record_Data_skinStretch,
xStep, dryStrokingTest, skinStretchTest, noPositionRot, rotAngles,
force_zero, positiveLimit, negativeLimit, WITH_MAGLEV, WITH_TRAJ,
comedi_voltage, force_curr, desNormForce, thetaY, nextStep,
forceActualFiltered, forceActualCorrected, subjectNum, state, gel, loc,
test, stepNum, overwriteFilenames );
            }

            printf("\nDo you wish to [0] exit OR [1] continue to Step
21 ( Location 3 - Wait to ReDry ) OR [2] return to Test Menu ?\n");
            std::cin >> continueResponse;
            if ( continueResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( continueResponse == 2 ) goto testMenu;

      case 21:// Right Distal WAIT

            printf("Get ready to test Location 3 for the last time, but
please WAIT to begin...");
            stepNum = 21;

            currLoc = 3;
            loc = testLocationList[currLoc-1];

            secondsAtRinse = getSecondsAtRinse ( loc, year, month,
date, subjectNum, test );

            while ( secondsSinceRinse < goalTime )
            {
                getDateTimeStr ( year, month, date, hours, minutes,
seconds );
                secondsCurrent = (atoi(hours)*3600 + atoi(minutes)*60
+ atoi(seconds));

                secondsSinceRinse = secondsCurrent - secondsAtRinse;
                waitTime_loc1 = goalTime - (double)secondsSinceRinse;

                waitTimeSeconds_loc1 = int(waitTime_loc1) % 60;
                waitTime_loc1 = waitTime_loc1/60.0;
                waitTimeInt_loc1 = (int)waitTime_loc1;
                if ( waitTimeOld_loc1 != waitTime_loc1 )
                {
                      printf("\nPlease wait to begin Step
22\tLocation 3 - ReDry\t%02d:%02d
",waitTimeInt_loc1,waitTimeSeconds_loc1);
```

```
                        waitTimeOld_loc1=waitTime_loc1;
                  }
            }

            secondsAtRinse = 0;
            secondsSinceRinse = 0;
            secondsCurrent = 0;
            minutesSinceRinse = 0;

            printf("\nThe 20 minute wait is complete.");

            printf("\nDo you wish to [0] exit OR [1] continue to Step
22 ( Location 3 - ReDry ) OR [2] return to Test Menu ?\n");
            std::cin >> continueResponse;
            if ( continueResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( continueResponse == 2 ) goto testMenu;

      case 22:// Right Distal ReDry 3,5

            stepNum = 22;

            currLoc = 3;
            stateNum = 5;
            gel = showerGelList[currGelOrder[currLoc-1]-1];// first gel
            loc = testLocationList[currLoc-1];// first test location
            state = moistureStateList[stateNum-1];// first state =
initial dry state

            printf("\nStep %i: %s - Location
%i\n",stepNum,state,currLoc);

            usleep(2000000);
            printf("\nPre force_zero: %f\n",force_zero[2]);
            retractAndZeroForceSensor ( device_handler, device,
cur_pos, nextStep, force_zero, comedi_voltage );
            printf("\nPost force_zero: %f\n",force_zero[2]);

            reDoForceDisplay22:
            forceDisplay ( device, comedi_voltage, force_curr,
ticks_per_millisecond, forceActualFiltered, force_zero, numSeconds );

            // Ask for user response, act accordingly
            loc22_dry:
            printf("\nStep 22 of 24... %s forearm, %s gel,
%s\n",loc,gel,state);
            printf("Do you wish to exit [0] OR continue [1] OR repeat
normal force printout [2] OR return to test menu [3] ?\n");
            std::cin >> userResponse;
            while ( userResponse != 0 && userResponse!=1 &&
userResponse!=2 && userResponse!=3) std::cin >> userResponse;
            if ( userResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( userResponse==2 ) goto reDoForceDisplay22;
            else if ( userResponse==3 )
            {
                  goto testMenu;
```

```
            }
            else if (userResponse != 0 && userResponse != 1 &&
userResponse != 2 && userResponse != 3 )
            {
                  printf(" \n--Incorrect Entry--\n");
                  goto loc22_dry;
            }
            else if ( userResponse==1 )
            {
                  DryStrokingSkinStretch ( device_handler, device,
cur_pos, cur_force, des_force, ticks_per_millisecond, time_next,
time_step, time_current_seconds, record_Data, record_Data_skinStretch,
xStep, dryStrokingTest, skinStretchTest, noPositionRot, rotAngles,
force_zero, positiveLimit, negativeLimit, WITH_MAGLEV, WITH_TRAJ,
comedi_voltage, force_curr, desNormForce, thetaY, nextStep,
forceActualFiltered, forceActualCorrected, subjectNum, state, gel, loc,
test, stepNum, overwriteFilenames );
            }

            printf("\nThe 20 minute wait is complete.");
            printf("\nDo you wish to [0] exit OR [1] continue to Step
23 ( Location 4 - Wait to ReDry ) OR [2] return to Test Menu ?\n");
            std::cin >> continueResponse;
            if ( continueResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( continueResponse == 2 ) goto testMenu;

      case 23:// Right Proximal WAIT

            printf("Get ready to test Location 4 for the last time, but
please WAIT to begin...");
            stepNum = 23;

            currLoc = 4;
            loc = testLocationList[currLoc-1];

            secondsAtRinse = getSecondsAtRinse ( loc, year, month,
date, subjectNum, test );

            while ( secondsSinceRinse < goalTime )
            {
                  getDateTimeStr ( year, month, date, hours, minutes,
seconds );
                  secondsCurrent = (atoi(hours)*3600 + atoi(minutes)*60
+ atoi(seconds));

                  secondsSinceRinse = secondsCurrent - secondsAtRinse;
                  waitTime_loc1 = goalTime - (double)secondsSinceRinse;

                  waitTimeSeconds_loc1 = int(waitTime_loc1) % 60;
                  waitTime_loc1 = waitTime_loc1/60.0;
                  waitTimeInt_loc1 = (int)waitTime_loc1;
                  if ( waitTimeOld_loc1 != waitTime_loc1 )
                  {
                        printf("\nPlease wait to begin Step
24\tLocation 4 - ReDry\t%02d:%02d
",waitTimeInt_loc1,waitTimeSeconds_loc1);
```

```
                        waitTimeOld_loc1=waitTime_loc1;
                  }
            }

            secondsAtRinse = 0;
            secondsSinceRinse = 0;
            secondsCurrent = 0;
            minutesSinceRinse = 0;

            printf("\nThe 20 minute wait is complete.");

            printf("\nDo you wish to [0] exit OR [1] continue to Step
24 ( Location 4 - ReDry ) OR [2] return to Test Menu ?\n");
            std::cin >> continueResponse;
            if ( continueResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( continueResponse == 2 ) goto testMenu;

      case 24:// Right Proximal ReDry 4,5

            stepNum = 24;

            currLoc = 4;
            stateNum = 5;
            gel = showerGelList[currGelOrder[currLoc-1]-1];// first gel
            loc = testLocationList[currLoc-1];// first test location
            state = moistureStateList[stateNum-1];// first state =
initial dry state

            printf("\nStep %i: %s - Location
%i\n",stepNum,state,currLoc);

            usleep(2000000);
            printf("\nPre force_zero: %f\n",force_zero[2]);
            retractAndZeroForceSensor ( device_handler, device,
cur_pos, nextStep, force_zero, comedi_voltage );
            printf("\nPost force_zero: %f\n",force_zero[2]);

            reDoForceDisplay24:
            forceDisplay ( device, comedi_voltage, force_curr,
ticks_per_millisecond, forceActualFiltered, force_zero, numSeconds );

            // Ask for user response, act accordingly
            loc24_dry:
            printf("\nStep 24 of 24... %s forearm, %s gel,
%s\n",loc,gel,state);
            printf("Do you wish to exit [0] OR continue [1] OR repeat
normal force printout [2] OR return to test menu [3] ?\n");
            std::cin >> userResponse;
            while ( userResponse != 0 && userResponse!=1 &&
userResponse!=2 && userResponse!=3) std::cin >> userResponse;
            if ( userResponse == 0 ) magLevExit ( device_handler,
des_force, mpGainSetType, gainVec );
            else if ( userResponse==2 ) goto reDoForceDisplay24;
            else if ( userResponse==3 )
            {
                  goto testMenu;
```

```
            }
            else if (userResponse != 0 && userResponse != 1 &&
userResponse != 2 && userResponse != 3 )
            {
                  printf(" \n--Incorrect Entry--\n");
                  goto loc24_dry;
            }
            else if ( userResponse==1 )
            {
                  DryStrokingSkinStretch ( device_handler, device,
cur_pos, cur_force, des_force, ticks_per_millisecond, time_next,
time_step, time_current_seconds, record_Data, record_Data_skinStretch,
xStep, dryStrokingTest, skinStretchTest, noPositionRot, rotAngles,
force_zero, positiveLimit, negativeLimit, WITH_MAGLEV, WITH_TRAJ,
comedi_voltage, force_curr, desNormForce, thetaY, nextStep,
forceActualFiltered, forceActualCorrected, subjectNum, state, gel, loc,
test, stepNum, overwriteFilenames );
            }

            printf("\nAll tests are now complete!\n\n");

            break;

      default: magLevExit ( device_handler, des_force, mpGainSetType,
gainVec ); break;
}

 //Close the comedi
      comedi_close(device);
 //Turn off the maglev
      if ( WITH_MAGLEV )
      {
            MaglevTurnOff ( device_handler );
            delete device_handler;
      }
      return 1;
} // end main
```

REFERENCES

[1] P. F. D. Naylor, "The skin surface and friction," *British J. of Dermatology,* vol. 67, pp. 239-248, Jul. 1955.

[2] S. Comaish and E. V. A. Bottoms, "The skin and friction: deviations from Amonton's Laws, and the effects of hydration and lubrication," *British J. of Dermatology,* vol. 84, pp. 37-43, Jan. 1971.

[3] D. R. Highley, *et al.*, "Frictional properties of skin," *J Investig Dermatol,* vol. 69, pp. 303-305, Sep. 1977.

[4] A. F. El-Shimi, "In vivo skin friction measurements," *J. of the Soc. of Cosmetic Chemists,* vol. 28, pp. 37-51, Feb. 1977.

[5] S. Weinstein, "New methods for the in-vivo assessment of skin smoothness and softness," *J. of the Soc. of Cosmetic Chemists,* vol. 29, pp. 99-115, Mar. 1978.

[6] S. Nacht, *et al.*, "Skin friction coefficient: changes induced by skin hydration and emollient application and correlation with perceived skin feel," *J. of the Soc. of Cosmetic Chemists,* vol. 32, pp. 55-65, Mar./Apr. 1981.

[7] L. J. Wolfram, "Friction of skin," *J. of the Soc. of Cosmetic Chemists,* vol. 34, pp. 465-476, Dec. 1983.

[8] M. Zhang and A. F. T. Mak, "In vivo friction properties of human skin," *Prosthetics and Orthotics Int.,* vol. 23, pp. 135-141, Aug. 1999.

[9] Koudine, *et al.*, "Frictional properties of skin: proposal of a new approach," *Int. J. of Cosmetic Science,* vol. 22, pp. 11-20, Feb. 2000.

[10] J. Asserin, *et al.*, "Measurement of the friction coefficient of the human skin in vivo: quantification of the cutaneous smoothness," *Colloids and Surfaces B (Biointerfaces),* vol. 19, pp. 1-12, Nov. 2000.

[11] N. Gitis and R. Sivamani, "Tribometrology of skin," *Tribology Trans.*, vol. 47, pp. 461-469, Oct. 2004.

[12] A. Ramalho, *et al.*, "In vivo friction study of human skin: Influence of moisturizers on different anatomical sites," *Wear,* vol. 263, pp. 1044-1049, Sept. 2007.

[13] C. Pailler-Mattei, *et al.*, "Contribution of stratum corneum in determining bio-tribological properties of the human skin," *Wear,* vol. 263, pp. 1038-1043, Sept. 2007.

[14] X. Liu, *et al.*, "Quantifying touch-feel perception: Tribological aspects," *Measurement Science and Technology,* vol. 19, Aug. 2008.

[15] M. Kwiatkowska, *et al.*, "Friction and deformation behaviour of human skin," *Wear,* vol. 267, pp. 1264-1273, Jun. 2009.

[16] H. Zahouani, *et al.*, "Characterization of the mechanical properties of a dermal equivalent compared with human skinin vivoby indentation and static friction tests," *Skin Research and Technology,* vol. 15, pp. 68-76, Feb. 2009.

[17] H. Zahouani, *et al.*, "Friction noise of human skin in vivo," *Wear,* vol. 267, pp. 1274-80, Jun. 2009.

[18] C. Pailler-Mattei and H. Zahouani, "Study of adhesion forces and mechanical properties of human skin in vivo," *J. of Adhesion Science and Technology,* vol. 18, pp. 1739-58, 2004.

[19] C. Pailler-Mattei and H. Zahouani, "Analysis of adhesive behaviour of human skin in vivo by an indentation test," *Tribology Int.*, vol. 39, pp. 12-21, Jan. 2006.

[20] A. Delalleau, *et al.*, "Characterization of the mechanical properties of skin by inverse analysis combined with the indentation test," *J. of Biomechanics,* vol. 39, pp. 1603-1610, 2006.

[21] G. Boyer, *et al.*, "In vivo characterization of viscoelastic properties of human skin using dynamic micro-indentation," in *Engineering in Medicine and Biology Soc., 2007. EMBS 2007. 29th Annual International Conference of the IEEE*, Aug. 2007, pp. 4584-4587.

[22] C. Pailler-Mattei, *et al.*, "In vivo measurements of the elastic mechanical properties of human skin by indentation tests," *Medical Engineering and Physics*, vol. 30, pp. 599-606, Jun. 2008.

[23] P. G. Agache, *et al.*, "Rigal. Mechanical properties and young's modulus of human skin in vivo," *Archives of Dermatological Research,* pp. 221--232, 1980.

[24] T. M. Kajs and V. Gartstein, "Review of the instrumental assessment of skin: Effects of cleansing products," *J. of the Soc. of Cosmetic Chemists,* vol. 42, pp. 249-271, Jul./Aug. 1991.

[25] F. M. Hendriks, *et al.*, "A numerical-experimental method to characterize the non-linear mechanical behaviour of human skin," *Skin Research and Technology*, vol. 9, pp. 274-283, Aug. 2003.

[26] F. M. Hendriks, *et al.*, "Influence of hydration and experimental length scale on the mechanical response of human skin in vivo, using optical coherence tomography," *Skin Research and Technology,* vol. 10, pp. 231-241, Nov. 2004.

[27] Siciliano, *et al.*, *Robotics: Modeling, Planning and Control*: Springer, 2011.

[28] J. D'Errico. (2009). *SLM - Shape Language Modeling* [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/24443-slm-shape-language-modeling

[29] S. K. Dogra. (2012 Sep.). *Mean Shifting* [Online]. Available: http://www.qsarworld.com/qsar-statistics-mean-shifting.php